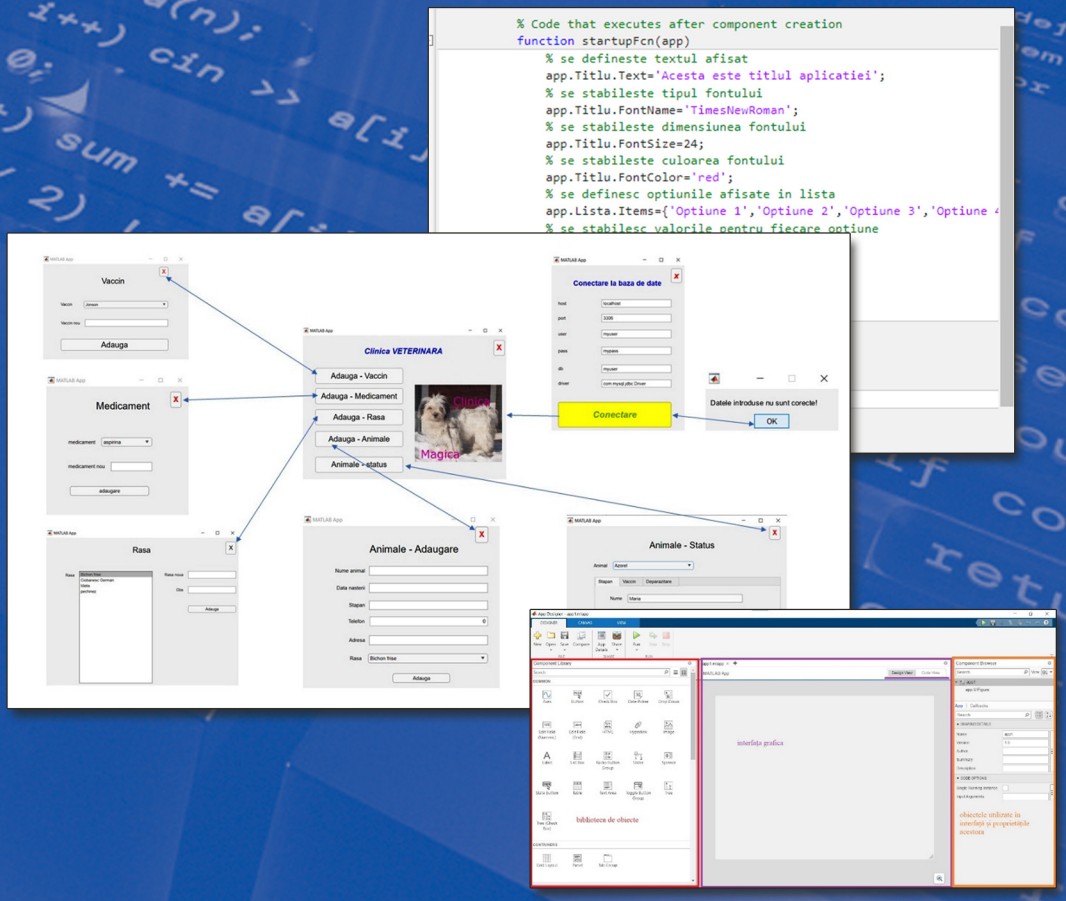
LUCIAN MARIUS RUSU

PROGRAMARE ÎN MATLAB



Editura POLITEHNICA

Colecţia "PROGRAMARE"

PROGRAMARE ÎN MATLAB

Cartea are ca scop prezentarea modului de lucru cu platforma Matlab. Programarea în mediul Matlab este esențială pentru toți cei care doresc implementarea algoritmilor pentru rezolvarea numerică a unor probleme inginerești și nu numai. Acest volum prezintă comenzile cele mai uzuale necesare pentru dezvoltarea aplicațiilor inginerești. Noțiunile de bază sunt introduse prin exemple ușor de înțeles și aplicat. De asemenea, aplicațiile rezolvate sunt de mare interes pentru studenții din domeniul ingineresc, dar și pentru toți cei care doresc să se familiarizeze cu aceste noțiuni.

Material bogat în conținut, reprezintă o sursă de documentare foarte utilă pentru cei ce doresc să dezvolte aplicații cu interfețe grafice utilizatorilor din mediul Matlab.

Referent ştiinţific: Prof. dr. ing. Lăcrămioara STOICU - TIVADAR

Conținutul se adresează inginerilor și tehnicienilor interesați în formarea abilitaților de operare cu ajutorul mediului MATLAB, pentru realizarea aplicațiilor care vizează măsurarea experimentală, prelucrarea datelor, crearea și gestionarea bazelor de date și alte probleme de interes în inginerie. Volumul este organizat în două pârți distincte. În prima parte sunt prezentate noțiuni teoretice, ilustrate ulterior în partea a doua prin dezvoltarea unor aplicații numerice în MATLAB.

Materialul, succint, dar bogat în conținut, se bucură de toate atributele unui manual introductiv valoros.

Referent ştiinţific: Prof. dr. ing. Arjana DAVIDESCU

LUCIAN MARIUS RUSU

PROGRAMARE ÎN MATLAB

Colecţia "PROGRAMARE"

EDITURA POLITEHNICA
TIMIŞOARA – 2023

EDITURA POLITEHNICA
Bd. Republicii nr. 9
300159 Timişoara, România

Tel. 0256.403.822
E-mail: editura@upt.ro

Redactor: Claudia MIHALI

ISBN 978-606-35-0520-1

CUPRINS

PREFAȚĂ

Odată cu evoluția tehnologiei s-au dezvoltat și programele de calcul matematic. La momentul actual sunt numeroase programe dedicate calculelor matematice, unul dintre cele mai utilizate fiind programul Matlab.

Programul Matlab este produs de compania MathWorks și este o aplicație ce poate fi utilizată în diferite domeni și conține funcții dedicate pentru foarte multe ramuri tehnice și nu numai. Vastele librării puse la dispoziție de acest program facilitează implementarea diferiților algoritmi tehnici.

Cartea este structurată pe trei capitole. In primul capitol este descrisă structura, caracteristicile și facilitațile programului Matlab.

Capitolul al doilea prezintă instrucțiunile și funcțiile generale ale programului Matlab. În acest capitol sunt prezentate diferitele tipuri de variabile și modul de accesare a informațiilor stocate în aceste variabile. De asemenea, sunt prezentate sintaxele celor mai uzuale comenzi puse la dispozitie de acest program. Pentru a facilita întelegerea sintaxei și modul de utilizare a instrucțiunilor s-au utilizat diferite exemple simple care să evidențieze caracteristicile fiecărei instrucțiuni.

În cel de al treilea capitol este exemplificat modul de realizare a diferitelor programe și aplicații utilizând programul Matlab. Acest capitol cuprinde zece exemple care utilizează diferite instrucțiuni și librări pentru rezolvarea unor probleme.

Această carte se adresează studenților din învățământul tehnic, care doresc să dobândească noțiunile de bază pentru realizarea de aplicatii matematice utilizând programul Matlab.

Doresc să mulțumesc doamnei prof.dr.ing. Lăcrămioara STOICU – TIVADAR și doamnei prof.dr.ing. Arjana DAVIDESCU pentru sprijinul acordat, precum și tuturor celor care m-au sustinut și au făcut posibilă realizarea acestei cărti.

1 INTRODUCERE

Matlab este un mediu de programare performat dedicat calculului numeric și reprezentărilor grafice din domeniul științific și ingineresc. [1]. Pagina oficială a programului Matlab este: https://www.mathworks.com/products/matlab.html. La această adresă, prin crearea unui cont, se poate descărca kitul de instalare pentru programul Matlab. De asemenea tot la aceasta adresă se găsește o documentație bogată despre comenzile, modulele și instrumentele utilizate în acest mediu de programare.

Numele „Matlab" provine de la **Mat**rix **Lab**oratory. Programul Matlab combina un mediu de lucru intuitiv cu un limbaj de programare orientat pe matrice și tablouri. Toate instrumentele dezvoltate sunt riguros testate și documentate în detaliu [2].

Dintre capabilitățile mediului de programare Matlab enumeram următoarele:

- utilizarea datelor experimentale complexe, de dimensiuni mari, dintr-o diversitatea de domenii;
- reprezentarea datelor în diverse moduri grafice sau tabelare;
- dezvoltarea de programe (script), funcții, clase, etc;
- creare de aplicații locale sau web;
- posibilitatea de utilizare a unor limbaje de programare externe (Python, C/C++, Fortran);
- conectarea la diferite componente hardware;
- posibilitatea utilizării calculului paralel (rezolvarea problemelor complexe pe mai multe calculatoare);
- utilizarea mediului "Cloud" pentru accesul de oriunde și partajarea facilă a rezultatelor.

Așadar mediul de programare Matlab este un instrument puternic de calcul numeric, care pune la dispoziția utilizatorului numeroase funcții care facilitează implementarea diferiților algoritmi de calcul al problemelor inginerești complexe.

Mediul de programare Matlab este disponibil atât în varianta locală (standalone) cât și în varianta online (accesul din browser). În continuare exemplificarea modului de lucru cu mediul Matlab se va realiza utilizând

versiunea academica R2022b (9.13.0.204977) pe 64 bit, apărută în 24 august 2022.

Interfața grafică inițială este prezentată în Fig. 1.1. Aceasta poate fi formatată funcție de prefetințele utilizatorului.

Ferestrele inițiale ce apar în interfață sunt:

- fereastra de comenzi (command window) unde se pot introduce comenzile și vizualiza rezultatele;
- fereastra de afișare a variabilelor (workspace) unde sunt afișate numele și tipul variabilelor declarate;
- fereastra directorului curent (current folder) unde sunt afișate fișierele din directorul curent;
- calea directorului curent;
- bara de comenzi unde se găsesc butoanele pentru operațiile necesare elaborării programului.

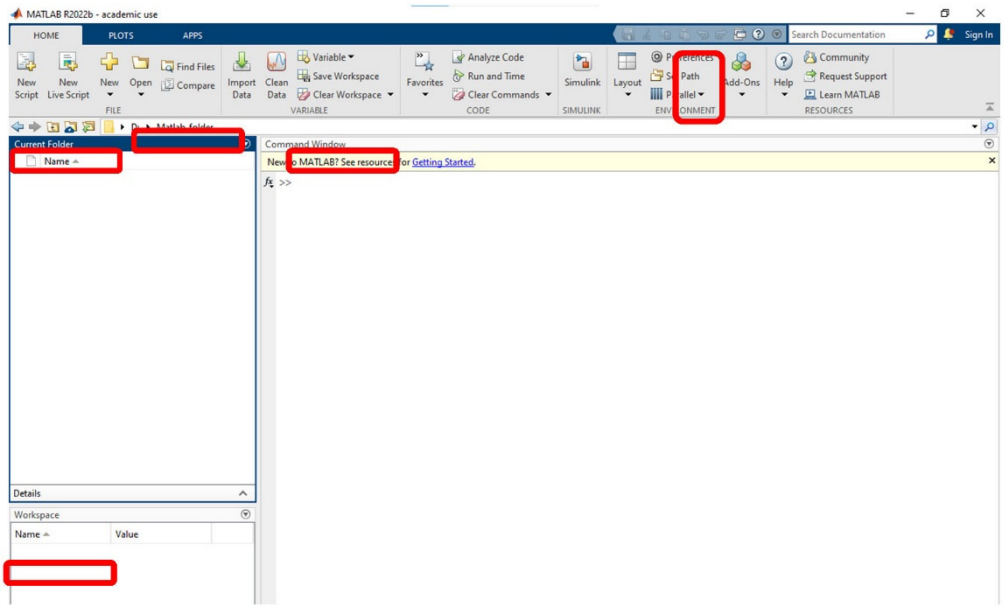


Fig 1.1. Interfața mediului de programare Matlab

Prin accesarea butonului „Layout” se poate modifica formatul interfeței grafice după dorința utilizatorului. Astfel pot fi adăugate, eliminate sau modificate ferestrele interfeței grafice. De asemenea fiecare fereastă poate fi redimensionată cu ajutorul mouse-ului.

2 FUNCȚII ȘI COMENZI DE BAZĂ ÎN MATLAB

Interacțiunea utilizatorului cu programul Matlab se realizează prin intermediul comenzilor și funcțiilor puse la dispoziție de program. Utilizatorul poate folosi fereastra de comenzi (command window) unde se rulează fiecare comanda în momentul introducerii, sau prin crearea unui script (unui fișier de tip *.m) unde comenzile introduse se execută doar în momentul apăsări butonului de execuție (run).

2.1 Funcții generale

Pentru a facilita realizarea unei aplicați în Matlab, există mai multe funcții și comenzi implementate de dezvoltator. În Tabelul 2.1 sunt prezentate câteva funcții de interes general din Matlab.

Tabelul 2.1. Funcții generale

Sintaxa	Descriere
help denumire_comanda	prin rularea acestei funcții în fereastra de comenzi se va afișa documentația referitoare la comanda menționată ca **denumire_comanda**
who	afișează numele variabilelor utilizate
whos	furnizează informații suplimentare referitoare la variabilele utilizate (nume, dimensiune, tip, etc)
clear var1 var2 ….	șterge din memorie variabilele var1, var2,…. dacă nu este menționată nici o variabilă atunci șterge toate variabilele
clc	curăță fereastra de comenzi
disp(variable)	afișează în command window valoarea variabilei
format optiune	stabilește formatul de afișare al numerelor pe ecran

Funcția format nu modifică formatul, tipul sau valoarea unei variabile numerice, doar schimbă modul de afișare pe ecran a numărului. Cu ajutorul acestei funcții putem formata modul de vizualizare a valorilor numerice astfel încât rezultatele afișate să fie ușor de înteles. Opțiunile acestei funcții sunt prezentate în Tabelul 2.2.

Tabelul 2.2. – Opțiuni funcție format

Optiune	Rezultat	Exemplu – constanta π
short	4 zecimale – reprezentare simplă	3.1416
long	15 zecimale – reprezentare simplă	3.141592653589793
shorte	4 zecimale – reprezentare științifică cu 2 digiți la putere	3.1416e+00
longe	15 zecimale – reprezentare științifică cu 2 digiți la putere	3.141592653589793e+00
shortg	4 zecimale – reprezentarea cea mai compactă dintre simplă sau științifică	3.1416
longg	15 zecimale – reprezentarea cea mai compactă dintre simplă sau științifică	3.141592653589793
shorteng	4 zecimale – reprezentare inginerească cu 3 digiți la putere	3.1416e+000
longeng	15 zecimale – reprezentare inginerească cu 3 digiți la putere	3.14159265358979e+000
hex	reprezentare hexazecimală	400921fb54442d18
+	afisează simbolul + pentru valori numerice pozitive, simbolul – pentru valori negative și spațiu pentru zero	+
bank	2 zecimale	3.14
rational	raport	355/113

În programul Matlab, nu este necesară declararea prealabilă a numelui sau tipul variabilei. Definirea unei variabile se poate realiza în orice moment. Pentru a defini o variabilă trebuie să i se atribuie o valoare, acest lucru realizându-se prin utilizarea simbolului egal (=).

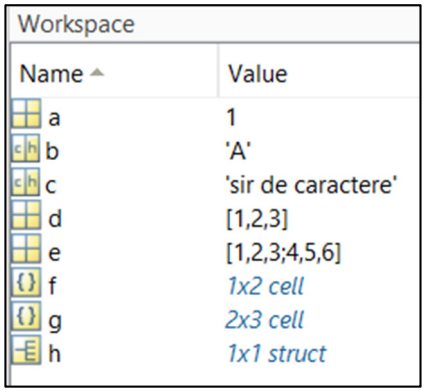
nume_variabila=valoare;

Numele variabilei este un sir de caractere. Primul caracter este obligatoriu sa fie alfanumeric, urmat de orice alte caractere exceptând caracterele speciale (exemple de caractere speciale: + - / * = & ^ % $ # @ ! ? ”). Caractere speciale de obicei au o anumită însemnătate în Matlab (operator, separator, comentariu, etc), iar în momentul execuției comenzi, programul încearcă sa realizeze operația, rezultând în cele mai multe situații o eroare. Singurul caracter special permis este underscore (_). De asemenea exista cuvinte rezervate care nu trebuie date ca nume variabilelor. Cuvintele cheie pot fi aflate cu comanda **iskeyword**. În Matlab 2022b cuvintele cheie sunt: 'break', 'case', 'catch', 'classdef', 'continue', 'else', 'elseif', 'end', 'for', 'function', 'global', 'if', 'otherwise', 'parfor', 'persistent', 'return', 'spmd', 'switch', 'try', 'while'.

În urma atribuirii unei valori, tipul variabilei poate fi: numeric, caracter, matrice, celula, structura, dată, timp, ș.a. În următoarele linii de comandă (Fig. 2.1) se exemplifică definirea mai multor tipuri de variabile.

```
a=1;
b='A';
c='sir de caractere';
d=[1,2,3];
e=[1,2,3;4,5,6];
f={1,'A'};
g={'B',5,'C';'Ion',[1,2],{1,'A'}};
h.N_1a=5;
h.N_1b.N_2a=[1,2,3];
h.N_1b.N_2b=f;
h.N_1b.N_2c=e;
```

a) Linii de comndă



b) Fereastra variabile

Fig. 2.1. Declararea variabilelor

Nu este necesar să definim noi tipul de variabilă, acest lucru se realizează automat de către program în momentul atribuirii unei valori. De asemenea unui nume de variabilă i se poate schimba tipul prin atribuiri succesive de valori de diferite tipuri. Variabilele numerice pot fi de mai multe feluri (Tabelul 2.3). Pentru a defini o variabilă de tip numeric, numelui acestei variabile i se va atribui o valoare numerică.

Tabelul 2.3. Variabile Numerice

Tip variabilă	Caracteristica
double	precizie dublă
single	precizie simplă
int8	întreg cu semn pe 8-bit
int16	întreg cu semn pe 16-bit
int32	întreg cu semn pe 32-bit
int64	întreg cu semn pe 64-bit
uint8	întreg fără semn pe 8-bit
uint16	întreg fără semn pe 16-bit
uint32	întreg fără semn pe 32-bit
uint64	întreg fără semn pe 64-bit

Variabilele de tip caracter se definesc utilizând caracterul special apostrof (') atât la începutul cât și la finalul șirului de caractere care îl atribuim variabilei.

Până în acest moment am discutat de variabile care conțin o valoare. Dacă dorim să creăm o variabilă care să conțină mai multe valori trebuie sa creăm un tabel. Astfel dacă valorile noastre sunt numerice atunci v-om crea o matrice sau un vector (în acest context, un vector reprezintă o matrice cu o singură linie sau coloana). Pentru a crea o matrice se utilizează paranteza pătrată ([ ]). Între parantezele pătrate se vor completa valorile numerice pe linie, separate între ele de simbolul virgulă (**,**) (se poate utiliza și spațiul ca separator dar nu este recomandat), iar pentru trecerea la linia următoare se utilizează simbolul punct și virgulă (**;**).

Dacă valorile pe care dorim sa le stocăm în tabel nu sunt toate de tip numeric atunci va trebui să utilizăm un tabel de celule (cell array). Definirea unui tabel de celule se face cu regulile precizate la matrice, diferența fiind utilizarea acoladelor ({})în locul parantezelor pătrate.

În următorul exemplu (Fig. 2.2) se generează matricea "**e**" cu 2 linii și 3 coloane și tabelul de celule "**f**" de asemenea cu 2 linii și 3 coloane dar unde celulele sunt de diferite tipuri.

```
e=[1,2,3;4,5,6];
g={'B',5,'C';'Ion',[1,2],{1,'A'}};
```

```
e =

     1     2     3
     4     5     6
```

```
g =

  2×3 cell array

    {'B'  }    {[  5]}    {'C'     }
    {'Ion'}    {[1 2]}    {1×2 cell}
```

a) Comandă b) Rezultat

Fig. 2.2. Variabile tip tabel

De asemenea se pot defini și variabile de tip structura. Structura acestor variabile este arborescentă. Și tabelele (matrice, cell array) pot fi multidimensionale, dar pentru fiecare dimensiune elementele trebuie sa fie de aceeași dimensiune. În cazul variabilelor de tip structură pot exista oricâte nivele, iar fiecare nivel poate avea oricâte valori, de orice tip. Pentru exemplificare se va crea o variabilă cu denumirea **'h'** (Fig. 2.3) care va avea două nivele. Primul nivel

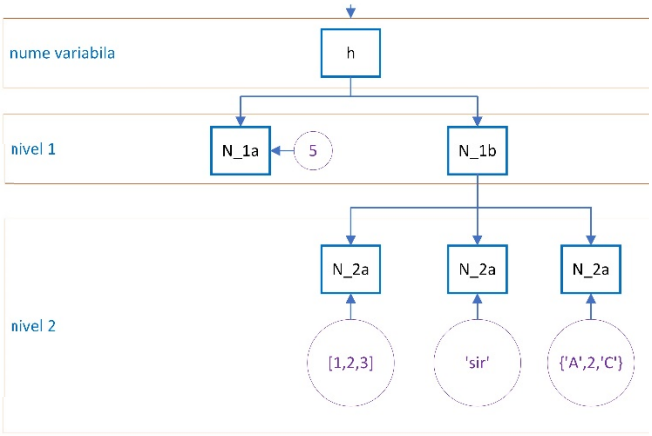
format din **N1_a** (numeric) și **N1_b** (structură), iar pe nivelul doi doar **N1_b** are valori și anume **N2_a** (vector)**, N2_b** (sir de caractere) respectiv **N2_c** (cell array).

```
h.N_1a=5;
h.N_1b.N_2a=[1,2,3];
h.N_1b.N_2b='sir';
h.N_1b.N_2c={'A',7,'C'};
```

```
h =

  struct with fields:

    N_1a: 5
    N_1b: [1×1 struct]
```



a)definirea variabilei b) nivelele variabilei

Fig. 2.3. Variabilă tip structură

Se observă că pe fiecare nivel conține alte tipuri de date.

Există unele variabile (constante) predefinite în Matlab. În Tabelul 2.4 sunt prezentate cele mai utilizate variabile predefinite în programul Matlab.

Tabelul 2.4. Variabile predefinite

Nume variabilă	Semnificație
ans	variabilă creată automat dacă rezultatul unei operați nu este atribuit niciunei variabile
pi	constanta π
Inf	reprezentarea lui ∞
NaN	reprezentarea lui 0/0 sau ∞/∞
i sau j	folosite la reprezentarea numerelor complexe ($\sqrt{-1}$)
clock	conține data și timpul curent
date	data curentă
eps	cel mai mic numar ce poate fi reprezentat în Matlab (2.2204 e-16)

Variabila **NaN** mai este utilizată și în cazul matricelor și a tabelelor de celule dacă lipseste valoarea unia dintre elementele acestora, deoarece nu se poate omite elementul respectiv.

2.2 Vectori și matrice

Programul Matlab fiind orientat pe calcul matriceal, cele mai multe facilități sunt pentru operațiile matriceale.

Definirea vectorilor se poate face prin introducere directă a valorilor pentru fiecare element, cum am precizat anterior, sau daca există o relație de legătură între elemente (un anumit increment) atunci vectorul poate fi definit astfel:

Nume_variabilă=[valoare_inițială : pas : valoare_finală];

În acest caz primul element al vectorului va fi **valoare_inițială**, iar următoarele elemente vor fi incrementate cu valoarea **pas**, până la ultimul element care va fi mai mic sau egal cu **valoare_finală**. Dacă pasul este omis atunci se va utiliza valoarea implicită care este unu. Dacă dorim o spațiere liniară a valorilor vectorului putem utiliza și comanda linspace.

Nume_variabilă=linspace(valoare_initială,valoare_finală,numar_elemente);

În acest caz vectorul va avea exact atâtea elemente câte sunt menționate în **numar_elemente**, valorile fiind egal spațiate între **valoare_inițială** și **valoare_finală**.

```
V1=[0:0.5:3];
V2=0:0.5:3;
V3=0:7;
V4=1:0.2:2.1;
V5=linspace(1,4,7);
```

a) comandă

```
V1 =
         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000

V2 =
         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000

V3 =
     0     1     2     3     4     5     6     7

V4 =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000

V5 =
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000
```

b) rezultat

Fig. 2.4. Declararea vectorilor

Se observă (Fig. 2.4) că vectori astfel definiți sunt de tip linie (adică o matrice cu o singură linie). La acest mod de definire se observă că se pot omite parantezele pătrate, vectorul V1 si V2 fiind identici. Dacă se omite pasul observam că implicit el este unu (vectorul V3). De asemenea observăm în cazul vectorului V4 că deși valoarea finală este 2.1, ultimul element al vectorului V4 este 2. Acest lucru se datorează pasului de 0.2. Astfel următorul element ar fi trebuit să fie 2+0.2=2.2 care este mai mare decât 2.1, ceea ce contravine ipotezei. În cazul vectorului V5 care este definit prin comanda linspace, pasul este calculat automat astfel încât primul element să aibă **valoare_ initială** (1) și ultimul element **valoare_finală** (4), iar vectorul să aibă lungimea menționată la **numarul_elemente** (7).

Pentru a selecta anumite elemente dintr-un vector sau matrice există mai multe posibilități prezentate în Tabelul 2.5.

Se va defini un vector **V** și o matrice **M**, iar parametrii **i, j,k,l,L,C** utilizați pentru acest exemplu sunt numere naturale.

```
V=[1,7,2,5,6,3,9,8,4,10];
M=[1, 2, 3, 4, 5; 6, 7, 8, 9, 10; 11, 12, 13, 14, 15; 16, 17, 18, 19, 20;…
    21, 22, 23, 24, 25; 26, 27, 28, 29, 30];
```

Tabelul 2.5. Selectarea elementelor dintr-un vector sau matrice

Sintaxă	Rezultat	Exemplu
V(i)	se selectează elementul de la poziția i	V(3) ans = 2
V(i:j)	se selectează elementele de la poziția i până la poziția j	V(2:5) ans =7 2 5 6
V(i:k:j)	se selectează elementele de la poziția i până la poziția j din k în k elemente (adică elementele de la pozițiile: i,i+k,i+2k,i+3k,……,j)	V(3:2:9) V(3:2:8) ans = 2 6 9 4 ans = 2 6 9
V([i1,i2,i3,…,in])	se selectează elementele de pe pozițiile i1,i2,i3,…,in	V([2,5,7]) ans = 7 6 9
V(:)	se selectează toate elementele vectorului	V(:) ans = 1 7 2 5 6 3 9 8 4 10

M(L,C)	se selectează elementul de pe linia L şi coloana C	M(2,5) ans =10
M(:,C)	se selectează coloana C	M(:,2) ans = 2 7 12 17 22 27
M(:,C1:C2)	se selectează coloanele de la C1 până la C2	M(:,2:4) ans = 2 3 4 7 8 9 12 13 14 17 18 19 22 23 24 27 28 29
M(:,C1:k:C2)	se selectează coloanele de la C1 până la C2 cu un pas k (coloanele C1,C1+k,C1+2k,….C2)	M(:,1:2:5) ans = 1 3 5 6 8 10 11 13 15 16 18 20 21 23 25 26 28 30
M(:,[C1,C2,…Cn])	se selectează coloanele C1,C2,…Cn	M(:,[2,4,5]) ans = 2 4 5 7 9 10 12 14 15 17 19 20 22 24 25 27 29 30
M(L,:)	se selectează linia L	M(3,:) ans = 11 12 13 14 15
M(L1:L2,:)	se selectează liniile de la L1 până la L2	M(3:5,:) ans = 11 12 13 14 15

		16 17 18 19 20 21 22 23 24 25
M(L1:k:L2,:)	se selectează liniile de la L1 până la L2 cu un pas k (liniile L1,L1+k,L1+2k,….L2)	M(1:2:5,:) ans = 1 2 3 4 5 11 12 13 14 15 21 22 23 24 25
M([L1,L2,…,Ln],:)	se selectează liniile L1,L2,….Ln	M([4,2,6],:) ans = 16 17 18 19 20 6 7 8 9 10 26 27 28 29 30
M(i:j,k:l)	se extrage submatricea aflată între liniile de la i până la j și coloanele de la k până la l	M(2:4,3:5) ans = 8 9 10 13 14 15 18 19 20
M([L1,L2,…,Ln], [C1,C2,…Cn])	se selectează elementele de la intersecțiile liniilor L1,L2,…,Ln cu coloanele C1,C2,…,Cn)	M([4,2,6],[2,4,5]) ans = 17 19 20 7 9 10 27 29 30
M(:,:)	se selectează întreaga matrice	M(:,:) ans = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

O facilitate a programului Matlab este aceea că o matrice poate fi tratată ca și un vector. De exemplu elementul de pe poziția **n** într-o matrice este al **n**-lea numărat pe coloane. În matricea **M** enunțată anterior, elementul M(15) are valoarea 13. Acest lucru este util deoarece parcurgerea întregii matrice se poate realiza cu o singură comandă repetitivă.

De asemenea există funcții pentru generarea matricelor speciale (Tabelul 2.6) cum ar fi: matricea identitate, unitate, zero, aleatorie, diagonală, etc. În tabelul următor exemplificam utilizarea acestor funcții.

Tabelul 2.6. Matrice speciale

Sintaxă	Rezultat	Exemplu
[]	matricea M nu are nici un element	M=[] M = []
ones(n)	returnează o matrice de dimensiune n x n cu toate elementele egale cu unu	M=ones(3) M = 1 1 1 1 1 1 1 1 1
ones(m,n)	returnează o matrice de dimensiune m x n cu toate elementele egale cu unu	M=ones(2,4) M = 1 1 1 1 1 1 1 1
zeros(n)	returnează o matrice de dimensiune n x n cu toate elementele egale cu zero	M=zeros(2) M = 0 0 0 0
zeros(m,n)	returnează o matrice de dimensiune m x n cu toate elementele egale cu zero	M=zeros(3,2) M = 0 0 0 0 0 0
eye(n)	returnează o matrice identitate de dimensiune n x n	M=eye(3) M = 1 0 0 0 1 0 0 0 1
eye(m,n)	returnează o matrice identitate de dimensiune m x n	M=eye(2,4) M = 1 0 0 0 0 1 0 0
rand(n)	returnează o matrice de dimensiune n x n cu elemente generate aleatoriu cu distributie uniformă între 0 și 1	M=rand(3) M = 0.7922 0.0357 0.6787 0.9595 0.8491 0.7577

		0.6557 0.9340 0.7431
rand(m,n)	returnează o matrice de dimensiune m x n cu elemente generate aleatoriu cu distributie uniformă între 0 şi 1	M=rand(3,2) M = 0.9572 0.1419 0.4854 0.4218 0.8003 0.9157
diag(v)	returnează o matrice patrată diagonală cu elementele vectorului v pe diagonala principală	v = 1 2 3 4 diag(v) ans = 1 0 0 0 0 2 0 0 0 0 3 0 0 0 0 4
diag(v,k)	returnează o matrice pătrată diagonală, cu elementele vectorului v pe diagonala k deasupra diagonalei principale dacă k>0 respectiv sub diagonala principală dacă k<0. Restul elementelor fiind zero	v = 1 2 3 4 diag(v,1) ans = 0 1 0 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0 4 0 0 0 0 0 diag(v,-2) ans = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 2 0 0 0 0 0 0 3 0 0 0 0 0 0 4 0 0

În programul Matlab există mai multe funcţii (Tabelul 2.7) care ne permit să determinăm dimensiunea matricelor respectiv să realizăm diferite calcule matriceale. Pentru exemplificare vom folosi vectorul **V** şi matricele **M** şi **M1**.

V =[3, 5, 2, 9, 4, 8, 6]
M=[1,2,3,4;5,6,7,8;4,9,2,2]
M1=[1,4,0;8,15,3;1,9,2]

Tabelul 2.7. Funcții utilizate

Sintaxă	Rezultat	Exemplu
length(V)	returnează lungimea vectorului V	length(V) ans = 7
length(M)	returnează valoarea cea mai mare dintre numărul de linii și coloane	length(M) ans = 4
[l,c]=size(V)	returnează numarul de linii și coloane ale vectorului, astfel observând dacă vectorul este linie sau coloană	size(V) ans = 1 7
[l,c]=size(M)	returnează numarul de linii și coloane ale matricei	size(M) ans = 3 4
det(M)	calculează determinantul unei matrice pătrate	det(M1) ans = -49
inv(M)	calculează inversa unei matrice pătrate	inv(M1) ans = -0.0612 0.1633 -0.2449 0.2653 -0.0408 0.0612 -1.1633 0.1020 0.3469
sum(V)	calculează suma elementelor unui vector	sum(V) ans = 37
sum(M)	calculează suma elementelor pentru fiecare coloană a matricei, rezultatele fiind stocate într-un vector	sum(M) ans = 10 17 12 14
prod(V)	calculează produsul elementelor unui vector	prod(V) ans = 51840
prod(M)	calculează produsul elementelor pentru fiecare coloană a matricei,	prod(M) ans = 20 108 42 64

	rezultatele fiind stocate într-un vector	
[m,p]=min(V)	determină minimul (m) și poziția (p) acestuia, dintre elementele unui vector	[m,p]=min(V) m = 2 p = 3
[m,p]=min(M)	determină minimul (m) și poziția (p) acestuia, dintre elementele fiecărei coloane a unei matrice. rezultatele sunt stocate în 2 vectori, unul (m) pentru minime și (p) unul pentru poziția acestuia în fiecare coloană	[m,p]=min(M) m = 1 2 2 2 p = 1 1 3 3
[m,p]=max(V)	determină maximul (m) și poziția (p) acestuia, dintre elementele unui vector	[m,p]=max(V) m = 9 p = 4
[m,p]=max(M)	determină maximul (m) și poziția (p) acestuia, dintre elementele fiecărei coloane a unei matrice. rezultatele sunt stocate în 2 vectori, unul (m) pentru minime și (p) unul pentru poziția acestuia în fiecare coloană	[m,p]=max(M) m = 5 9 7 8 p = 2 3 2 2
mean(V)	calculează media aritmetică a elementelor unui vector	mean(V) ans = 5.2857
mean(M)	calculează media aritmetică a elementelor fiecărei coloane a unei matrice, rezultatele fiind stocate într-un vector	mean(M) ans = 3.3333 5.6667 4.0000 4.6667
geomean(V)	calculează media geometrică a elementelor unui vector	geomean(V) ans = 4.7155
geomean(M)	calculează media geometrică a elementelor fiecărei coloane a unei	geomean(M) ans =

	matrice, rezultatele fiind stocate într-un vector	2.7144 4.7622 3.4760 4.0000
harmmean(V)	calculează media armonică a elementelor unui vector	harmmean(V) ans = 4.1516
harmmean(M)	calculează media geometrică a elementelor fiecărei coloane a unei matrice, rezultatele fiind stocate într-un vector	harmmean(M) ans = 2.0690 3.8571 3.0732 3.4286

În matematică funcțiile trigonometrice reprezintă un capitol important. Astfel programul Matlab pune la dispozitia utilizatorului funcții pentru calculul trigonometric direct, invers, hiperbolic (Tabelul 2.8).

Tabelul 2.8. Funcții trigonometrice

Funcții trigonometrice	Sintaxă	Rezultat
Directe - radiani	sin()	calculează sinusul (argument în [rad])
	cos()	calculează cosinusul (argument în [rad])
	tan()	calculează tangenta (argument în [rad])
	cot()	calculează cotangenta (argument în [rad])
	sec()	calculează secanta (argument în [rad])
	csc()	calculează cosecanta (argument în [rad])
Directe - grade	sind()	calculează sinusul (argument în [grad])
	cosd()	calculează cosinusul (argument în [grad])
	tand()	calculează tangenta (argument în [grad])
	cotd()	calculează cotangenta (argument în [grad])
	secd()	calculează secanta (argument în [grad])
	cscd()	calculează cosecanta (argument în [grad])
Inverse	asin()	calculează arcsinusul
	acos()	calculează arccosinusul
	atan()	calculează arctangenta
	atan2()	calculează arctangenta dacă argumentul este complex
	acot()	calculează arccotangenta
	asec()	calculează arcsecanta
	acsc()	calculează arccosecanta
Hiperbolice	sinh()	calculează sinusul hiperbolic
	cosh()	calculează cosinusul hiperbolic

	tanh()	calculează tangenta hiperbolică
	coth()	calculează cotangenta hiperbilocă
	sech()	calculează secanta hiperbolică
	csch()	calculează cosecanta hiperbolică
Hiperbolice inverse	asinh()	calculează arcsinusul hiperbolic
	acosh()	calculează arccosinusul hiperbolic
	atanh()	calculează arctangenta hiperbolică
	acoth()	calculează arccotangenta hiperbolică
	asech()	calculează arcsecanta hiperbolică
	acsch()	calculează arccosecanta hiperbolică

Tabelul 2.9. Funcții matematice

Sintaxă	Rezultat	Exemplu
sqrt()	extrage radical de ordinul 2 (rădăcina pătrată)	sqrt(25) ans = 5
exp()	calculează exponențiala (puteri ale numarului e)	exp(3) ans = 20.0855
log()	calculează logaritmul natural (logaritm în baza e)	log(20.0855) ans = 3.0000
log2()	calculează logaritmul în baza 2	log2(2^5) ans = 5
log10()	calculează logaritmul zecimal	log10(10^7) ans = 7
pow2()	calculează puteri ale lui 2	pow2(10) ans = 1024
abs()	determină valoarea absolută (modulul)	abs(-13) ans = 13
conj()	calculează complex conjugatul	conj(2-3i) ans = 2.0000 + 3.0000i
real()	extrage partea reală	real(2-3i) ans = 2
imag()	extrage partea imaginară	imag(2-3i) ans = -3

În Tabelul 2.9 sunt prezentate funcțiile matematice elementare pentru ridicarea la putere, exponențială, logaritmică precum și utilizarea numerelor complexe.

Operatorii matematici disponibili în aplicația matlab sunt prezentați și exemplificați în Tabelul 2.10. Pentru exemplificare se vor utiliza matricele **M1**, **M2** și **M3** definite astfel:

```
M1=[1,2,3;4,5,6];
M2=[4,5,6;1,2,3];
M3=[2-3i,4+5i,7-3i;1+1i,2+2i,9-5i];
```

Tabelul 2.10. Operatori matematici

Operatori matematici	Rezultat	Exemplu
+	adunare	2+2 ans = 4
-	scădere	3-2 ans = 1
*	înmulțire	2*3 ans = 6
/	împărțire	5/2 ans = 2.5000
\	împărțire la stanga	5\2 ans = 0.4000
^	ridicare la putere	5^3 ans = 125
.*	înmulțirea între două matrice (sau vectori) element cu element	M1.*M2 ans = 4 10 18 4 10 18
./	împărțirea între două matrice (sau vectori) element cu element	M1./M2 ans = 0.2500 0.4000 0.5000 4.0000 2.5000 2.0000
.\	împărțirea la stânga între două matrice (sau vectori) element cu element	M1.\M2 ans = 4.0000 2.5000 2.0000 0.2500 0.4000 0.5000
.^	ridicarea la putere a unei matrice (sau vector) element cu element	M1.^3 ans = 1 8 27 64 125 216

’	transpunere și conjugare	M1' ans = 1 4 2 5 3 6 M3' ans = 2.0000 + 3.0000i 1.0000 - 1.0000i 4.0000 - 5.0000i 2.0000 - 2.0000i 7.0000 + 3.0000i 9.0000 + 5.0000i
.’	transpunere	M1.' ans = 1 4 2 5 3 6 M3.' ans = 2.0000 - 3.0000i 1.0000 + 1.0000i 4.0000 + 5.0000i 2.0000 + 2.0000i 7.0000 - 3.0000i 9.0000 - 5.0000i

2.3 Comenzi decizionale și repetitive

La fel ca în oricare limbaj de programare și programul Matlab conține instrucțiuni condiționale (decizionale) și repetitive. Instrucțiunile condiționale sunt **IF** și **SWITCH**, iar cele repetitive sunt **FOR** și **WHILE**. Fiecare din aceste instrucțiuni încep cu cuvântul cheie dedicat și se încheie cu cuvântul cheie **END**. La capătul liniilor de program care conțin cuvinte cheie nu se pune simbolul “**;**”.

Sintaxă:

```
if condiție_1
        bloc instrucțiuni 1;
elseif condiție_2
        bloc instrucțiuni 2;
else
        bloc instrucțiuni 3;
end
```

Instrucțiunea decizională **if** are sintaxa prezentată anterior. Dacă este îndeplinită **conditia_1** atunci se vor executa instrucțiunile din **bloc instrucțiuni**

1 apoi se va sări la linia de comandă **end** care va încheia comanda if. Dacă **condiție_1** nu este îndeplinită atunci se va trece la verificarea **condiției_2**. Dacă aceasta este îndeplinită atunci se va executa blocul de instrucțiuni 2, urmând apui să se sară la comanda **end**, care va încheia comanda if. Condiții suplimentare prin utilizarea cuvântului cheie **elseif** pot fi mai multe. Verificarea acestora se va face în ordinea în care au fost scrise. Dacă se îndeplinește una dintre condiții atunci se vor executa comenzile din blocul de instrucțiuni aferent. În acest moment se sare la comanda **end** pentru a încheia instrucțiunea IF, fără a mai verifica și restul condițiilor impuse prin **elseif** dacă acestea mai există. Dacă nici una dintre condiții nu a fost îndeplinită atunci se execută instrucțiunile din blocul de instrucțiuni 3.

Pentru exemplificare v-om realiza o conversie între o notă de la 0 la 10 la un calificativ de la A la F conform Tabelului 2.11.

Tabelul 2.11. Conversie note

Interval nota	Calificativ
[9;10]	A
[8;9)	B
[7;8)	C
[6;7)	D
[5;6)	E
[0;5)	F

Pentru conversia notelor s-a realizat următorul program (script)

```
% conversie nota - calificativ
clear; % se sterg toate variabilele
clc;   % se curata command window-ul
nota=input('Introduceti o notă intre 0 si 10. Nota= '); % se introduce de la
tastatura o valoare numerică
% Fiecarui interval declarat ii corespunde un calificativ
% Prin instruciunea if variabilei calificativ ii va fi atribuit valoarea
% corespunzatoare
if (nota>=9)
        calificativ='A';
        elseif (nota>=8) && (nota<9)
                calificativ='B';
        elseif (nota>=7) && (nota<8)
                calificativ='C';
        elseif (nota>=6) && (nota<7)
```

```
            calificativ='D';
        elseif (nota>=5) && (nota<6)
            calificativ='E';
        else
            calificativ='F';
end
disp('Calificativul pentru nota introdusa este:'); % afiseaza in CW
disp(calificativ); % se afiseaza calificativul
```

La rularea programului creat se cere introducerea unui valori numerice ce reprezintă nota, apoi programul afişează calificativul aferent notei introduse (Fig. 2.5)

```
Introduceti o notă intre 0 si 10. Nota= 7
Calificativul pentru nota introdusa este:
C
```

Fig. 2.5. Rularea programului pentru comanda if

O altă comandă condiţională este instrucţiunea **SWITCH.** Această comandă este una de tip selecţie şi are următoarea sintaxă:

Sintaxă:

```
switch operator
    case expresie
        bloc instructiuni 1;
    case {expresie1,expresie2,…..}
        bloc instructiuni 2;
    otherwise
        bloc instrucţiuni 3;
end
```

Instrucţiunea **switch** este utilizată când avem diferite opţiuni de selecţie. Astfel variabila **operator** este comparată cu **expresie** şi daca sunt identice atunci se execută blocul de instrucţiuni aferent. Dacă aceeaşi secvenţă de instrucţiuni trebuie executată pentru mai multe expresii atunci se poate crea o mulţime de expresii prin enumerarea acestora între acolade şi separate între ele prin virgulă. În cadrul acestei instrucţiuni pot exista oricâte cazuri (case). Expresiile se evaluează în ordinea în care au fost scrise. Când se găseşte o expresie validă atunci se execută blocul de instrucţiuni aferent, urmat ieşirea din instrucţiunea switch fără a mai verifica restul cazurilor. De asemenea există posibilitatea utilizării cuvântului cheie **otherwise** care ne permite executarea blocului de instrucţiuni 3 în cazul în care nici o opţiune nu este validă.

Pentru exemplificare s-a scris un program (script) care va citi de la tastatură un șir de caractere care va fi comparat cu ajutorul comenzi switch cu elementele scrise în lista de fructe respectiv legume.

```
sir=input('Introduceti un frunct sau o leguma  ','s');
switch sir
        case {'mar','para','pruna','portocala','banana'}
                disp ('ai introdus un fruct');
        case {'ardei','rosie','castravete','morcov'}
                disp('ai introdus o leguma')
        otherwise
                disp ('nu exista in lista')
end
```

Prin rularea de trei ori a programului (pentru a valida optiunile de selectie) am obtinut rezultatele (Fig. 2.6).

```
Introduceti un frunct sau o leguma  mar
ai introdus un fruct

Introduceti un frunct sau o leguma  morcov
ai introdus o leguma

Introduceti un frunct sau o leguma  avocado
nu exista in lista
```

Fig. 2.6. Rularea programului pentru comanda switch

Pentru a executa un bloc de comenzi de mai multe ori este necesar utilizarea unei instrucțiuni repetitive. În acest scop se pot utiliza instrucțiunile **for** dacă se dorește repetarea de un nr cunoscut de ori, sau instrucțiunea **while** care permite repetarea atâta timp cât o condiție este îndeplinită. Comanda **for** are următoarea sintaxă:

Sintaxă:

```
for variabila=expresie
        bloc instrucțiuni;
end
```

În cazul comenzi repetitive **for** avem o **variabila** care ia diferite valori. Pentru fiecare valoare a acesteia se execută comenzile din bloc **instrucțiuni**. Numărul de repetări este cunoscut apriori. În cadrul blocului de instrucțiuni se

poate utiliza valoarea variabilei folosită ca și contor sau nu. Pentru exemplificare, se dorește afisarea de 10 ori pe ecran a textului ”Hello world”, indexat (Fig. 2.7).

```
for i=1:10
variabila=horzcat(num2str(i),' -> Hello world!');
disp(variabila);
end
```

```
>> test
1 -> Hello world!
2 -> Hello world!
3 -> Hello world!
4 -> Hello world!
5 -> Hello world!
6 -> Hello world!
7 -> Hello world!
8 -> Hello world!
9 -> Hello world!
10 -> Hello world!
```

a) program b) rezultat

Fig. 2.7. Program exemplu pentru comanda for

Cealaltă instrucțiune repetitivă disponibilă în programul Matlab este while, care are următoarea sintaxă:

Sintaxă

```
while expresie
        bloc instrucțiuni;
end
```

Instrucțiunea **while** este tot una repetitivă, doar că în acest caz nu se cunoaște numărul de repetări. Adică blocul de instrucțiuni se execută atâta timp cât condiția **expresie** este adevărată. În cazul utilizării acestei comenzi trebuie avut grijă sa nu intrăm într-o buclă infinită.

Se doreste citirea de la tastatură a unui număr cuprins în intervalul [1;10]. Dacă utilizatorul introduce o valoare din afara intervalului atunci să se afișeze un mesaj de eroare și să se citească o alta valoare. Când valoarea introdusă aparține intervalului atunci să se afișeze aceasta valoare (Fig. 2.8).

```
var=input('Introduceti o valoare in intervalul [1;10]. var= ');
while (var<1)||(var>10)
        disp('Valoarea introdusa nu apartine intervalului [1;10]');
        var=input('Introduceti o valoare in intervalul [1;10]. var= ');
end
disp(var);
```

```
>> test
Introduceti o valoare in intervalul [1;10]. var= 12
Valoarea introdusa nu apartine intervalului [1;10]
Introduceti o valoare in intervalul [1;10]. var= 6
     6
```

Fig. 2.8. Rularea programului pentru comanda while

În cazul instrucțiunilor repetitive (for, while) există posibilitatea ieșirii forțate din buclă utilizând comanda **break**. La momentul întâlnirii acestei comenzi se iese automat din buclă fără a executa restul instrucțiunilor. Ieșirea din buclă în acest mod însă nu este recomandată.

Pentru a putea crea condițiile necesare instrucțiunilor decizionale sunt necesare utilizarea expresiilor și operatorilor logici. În Tabelul 2.12 sunt explicate expresiile logice iar în Tabelul 2.13 sunt prezentați operatorii logici.

Tabelul 2.12. Expresii logice

Expresii logice	Rezultat
a == b	returnează adevărat (true - 1) dacă a este egal cu b și fals (false - 0) dacă sunt diferite
a ~= b	returnează adevărat (true - 1) dacă a este diferit de b și fals (false - 0) dacă sunt egale
a < b	returnează adevărat (true - 1) dacă a este mai mic decât b și fals (false - 0) dacă a este mai mare sau egal cu b
a <= b	returnează adevărat (true - 1) dacă a este mai mic sau egal cu b și fals (false - 0) dacă a este mai mare decât b
a > b	returnează adevărat (true - 1) dacă a este mai mare decât b și fals (false - 0) dacă a este mai mic sau egal cu b
a >= b	returnează adevărat (true - 1) dacă a este mai mare sau egal cu b și fals (false - 0) dacă a este mai mic decât b
srtcmp(sir1,sir2)	verifică egalitatea între șiruri de caractere. dacă sir1 este identic cu sir2 atunci rezultatul este adevărat (true – 1) iar dacă șirurile sunt diferite rezultatul este fals (false -0)

Tabelul 2.13. Operatrori logici

Operatori logici	Rezultat
&	si logic
|	sau logic
~	negatie

Simbolurile operatorilor logici **şi (&)** şi **sau (|)** pot fi dublate pentru o procesare mai rapidă. În cazul expresiilor lungi cu mai mulţi operatori logici dacă utilizăm simbolurile dublate (**&&** sau ||) atunci când se poate trage o concluzie despre rezultatul final, se trece mai departe fără a calcula întreaga expresie.

2.4 Reprezentări grafice

O modalitate des utilizată în inginerie este prezentarea rezultatelor sub formă de grafic. În acest scop limbajul Matlab pune la dispoziţie mai multe funcţii pentru reprezentarea grafică sub diverse forme a rezultatelor. În continuare se vor prezenta cele mai uzuale funcţii pentru reprezentare grafică.

Pentru reprezentarea unui grafic 2D se va utiliza funcţia **plot**. În cadrul acestei comenzi variabilele **x** şi **y** reprezintă doi vectori de lungimi egale, iar variabila **format** precizează modul de formatare a punctelor şi a curbei (culoare, forma, etc).

plot (x,y,format);

Astfel, prin rularea acestei comenzi se va afişa într-o fereastră un grafic bidimensional, unde pentru fiecare din perechile de valori (x,y) se va afişa un punct pe grafic (acesta fiind motivul pentru care lungimea vectorilor x şi y trebuie să fie aceeaşi). Aceste puncte reprezentate grafic vor fi unite cu o linie. Pentru formatarea punctelor şi liniei ce le uneşte avem următoarele opţiuni (Tabelul 2.14).

Tabelul 2.14. Simboluri utilizate la formatarea graficului tip plot

Marcaj puncte		**Tip linie**	
.	punct	-	continuă
o	cerc	:	punctată
x	x	-.	linie punct
+	plus	--	linie întreruptă
*	stea	(nimic)	Fără linie
s	pătrat	**Culoare curbă**	
d	romb	b	albastru (blue)
v	triunghi cu vârful în jos	g	verde (green)
^	triunghi cu vârful în sus	r	roşu (red)
<	triunghi cu vârful în stânga	c	cian (cyan)
>	triunghi cu vârful în dreapta	m	magenta
p	pentagramă	y	galben (yellow)
h	hexagramă	k	negru (black)
		w	alb (white)

Pe lângă aceste formatări mai pot fi utilizare și alte formatări folosind cuvinte cheie (Tabelul 2.15). Acestea se referă la formatarea individuală a liniei sau a marcajelor punctelor. Nu contează ordinea în care sunt menționate cuvintele cheie.

Tabelul 2.15. Cuvinte cheie utilizate pentru formatare grafic

Cuvinte cheie	Rezultat
'LineWidth',val	grosimea curbei va fi de **val** pixeli
'MarkerEdgeColor',val	marginea marcajului punctului va avea culoarea **val** (se vor folosi simbolurile pentru culoarea curbei)
'MarkerFaceColor',val	interiorul marcajului punctului va avea culoarea **val** (se vor folosi simbolurile pentru culoarea curbei)
'Markersize',val	dimensiunea marcajului punctului va avea **val** pixeli

Pentru exemplificare vom definii urmatoarele perechi de numere: (1,28); (2,37); (3,41); (4,58); (5,60); (6,63); (7,66); (8,83); (9;89); (10,95) care reprezintă coordonatele punctelor ce se vor afișa grafic.

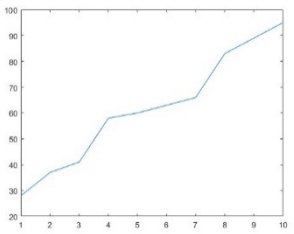
Definim vectorii x și y care vor avea coordonatele reprezentate pe abscisă respectiv pe ordonata.

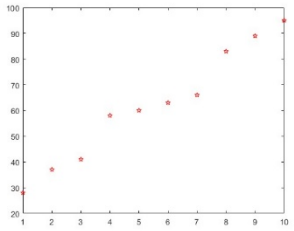
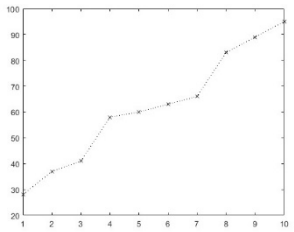
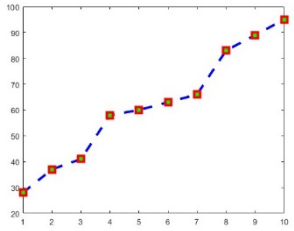
x =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

y =[28, 37, 41, 58, 60, 63, 66, 83, 89, 95];

În tabelul 2.16 se vor prezenta diferite formatări are reprezentării grafice a celor doi vectori utilizând funcția plot.

Tabelul 2.16. Reprezentări grafice

plot(x,y)	funcția plot poate fi rulată fără a menționa parametrii de formatare, în acest caz formatarea implicită presupune trasarea curbei cu culoare albastră fără a marca perechile de puncte	

plot(x,y,'pr')	parametrul de formatare **'pr'** presupune marcarea punctelor cu pentagrama de culoarea roşie fără a trasa curba	
plot(x,y,'xk:')	parametrul de formatare **'xk:'** presupune marcarea punctelor cu **x**, unindu-le cu linie punctată, atât punctele cât şi curba având culoarea neagra	
plot (x,y,'bs--',… 'LineWidth',3,… 'MarkerSize',10,… 'MarkerEdgeColor','r',… 'MarkerFaceColor','g')	parametrul de formatare **'bs--'** presupune marcarea punctelor cu **patrat**, unindu-le cu linie întreruptă de culoare albastră; 'LineWidth',3 generează curba cu o grosime de 3 pixeli 'MarkerSize',10 setează dimensiunea marcajelor punctelor la 10 pixeli 'MarkerEdgeColor','r' colorează muchia marcajului punctelor cu rosu 'MarkerFaceColor','g' colorează generează interiorul marcajului punctelor cu verde	

Pentru a marca corespunzător reprezentarea grafică a datelor, este necesar a eticheta axele, a da un titlu graficului precum şi al scala în anumite situaţii.

Programul Matlab pune la dispoziție funcții dedicare pentru a realiza acest lucru. Pentru etichetarea axelor se folosește comanda **xlabel** pentru abscisa și **ylabel** pentru ordonată. Titlul se afișează utilizând funcția **title**. Pentru aceste funcții de etichetare, după numele funcției se va trece între paranteze rotunde șirul de caractere ce se dorește afișat.

Limitele pentru axe se pot seta utilizând comenzile **xlim** sau **ylim** pentru a definii fie limitele abscisei fie ale ordonatei. Dacă se dorește scalarea ambelor axe, acest lucru se poate realiza utilizând ambele comenzi menționate anterior sau prin utilizarea comenzii **axis**.

Funcția **grid** ne permite afișarea sau eliminarea grilei prin cei doi parametri ai acesteia: **on** sau **off**.

De asemenea se poate suprapune pe grafic, într-o anumită pozitie, un text. Acest lucru se realizează cu funcția **text**. Din punctul de coordonate (Cx,Cy) se începe scrierea șirului de caractere stocat în variabila text.

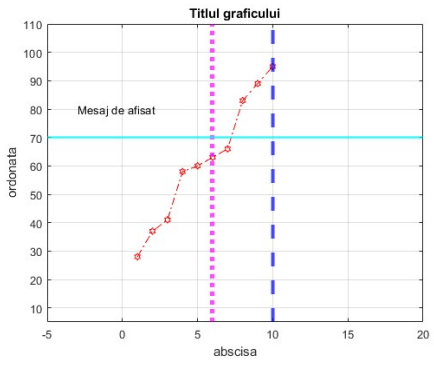
Sintaxa: text (Cx,Cy,text);

Pentru a marca pe grafic anumite valori (minime, maxime, medii, etc) uneori este util trasarea unor linii suplimentare. Pentru a trasa linii orizontale sau verticale există implementate funcțiile **xline** respectiv **yline** care vor trasa la o anumită valoare o linie. Si acest marcaj poate fi formatat în acelaşi fel cum se realizează formatarea curbei reprezentate grafic.

În Fig. 2.9 se va prezenta un exemplu în care vor fi exemplificate comenzile de formatare a graficului prezentate anterior.

```
x=[1:10];
y=[28,37,41,58,60,63,66,83,89,95];
plot(x,y,'hr-.');
title('Titlul graficului');
xlabel('abscisa');
ylabel('ordonata');
axis([-5,20,5,110]);
grid on;
text(-3,80,'Mesaj de afisat');
xline(6,'m:','LineWidth',4);
xline(10,'b--','LineWidth',3);
yline(70,'c-','LineWidth',2);
```

a) Linii de comandă



b) grafic

Fig. 2.9. Exemplu grafic 2D

Pentru suprapunere mai multor grafice în aceeași figură se poate utiliza comanda **hold**. Dacă aceasta este activă (hold on;) atunci orice reprezentare grafică va fi suprapusă peste cea anterioară. Aceste suprapuneri realizându-se până când funcția **hold** va fi dezactivată (hold off;). O altă modalitate de suprapunere a graficelor este de a utiliza o singură dată funcția plot, iar ca argumente să prezentăm toate valorile curbelor ce se doresc suprapuse. Fiecare curba poate fi formatată individual. Dacă se optează pentru formatarea automată a acestora, aplicația Matlab va schimba automat culoarea fiecărei curbe pentru a le putea distinge. În acest caz este necesară prezența unei legende pentru a eticheta curbele reprezentate. Acest lucru se poate realiza utilizând funcția **legend**. Argumentul acestei funcţii trebuie să fie un tablou de celule (cell array) care să conțină câte o etichetă pentru fiecare curbă reprezentată grafic, ordinea etichetelor trebuie sa fie aceeași cu ordinea de afișare a curbelor. Un alt argument util al funcției **legend** este poziția acesteia în raport cu graficul. În Tabelul 2.17 sunt prezentate posibilitățile de poziționare a legendei.

Tabelul 2.17. Poziția legendei

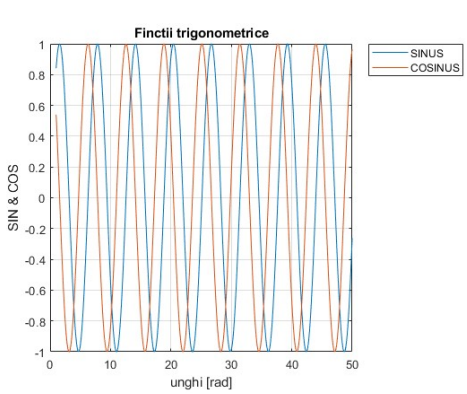
'north'	partea superioară în interior
'south'	partea inferioară în interior
'east'	partea dreaptă în interior
'west'	partea stângă în interior
'northeast'	partea superioară-dreapta în interior
'northwest'	partea superioară-stânga în interior
'southeast'	partea inferioară-dreapta în interior
'southwest'	partea inferioară-stânga în interior
'northoutside'	partea superioară în exterior
'southoutside'	partea inferioară în exterior
'eastoutside'	partea dreaptă în exterior
'westoutside'	partea stângă în exterior
'northeastoutside'	partea superioară-dreapta în exterior
'northwestoutside'	partea superioară-stânga în exterior
'southeastoutside'	partea inferioară-dreapta în exterior
'southwestoutside'	partea inferioară-stânga în exterior
'best'	plasare automată a legendei în interior cu minim de suprapunere peste grafic
'bestoutside'	plasarea automată a legendei în exterior în colțul dreapta sus dacă legenda este verticală sau dedesubtul axelor dacă legenda este orizontală

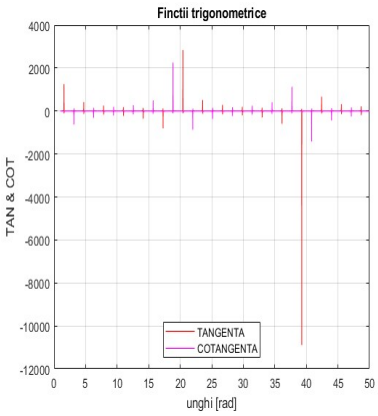
Pentru o mai bună înțelegere a reprezentărilor grafice se va rezolva următorul exercitiu:

Să se genereze două grafice, în care să se reprezinte funcțiile sinus și cosinus respectiv funcțiile tangentă și cotangentă. Graficele să conțină legenda și sa fie etichetate corespunzător.

Vectorul **x** va fi un vector ce are ca elemente numerele de la 1 la 50 cu un pas de 0.01. Pentru fiecare valoare din vectoul **x** se va calcula sinusul, cosinusul, tangenta și cotangenta iar rezultatele vor fi stocate în vectorii **y1**, **y2**, **y3** respectiv **y4**. Se vor folosi cele două metode prezentate pentru suprapunerea graficelor. Graficele obținute sunt prezentate în Fig. 2.10. Programul (scriptul) creat conține următoarele linii de comandă:

```
x=1:0.01:50;
y1=sin(x);
y2=cos(x);
y3=tan(x);
y4=cot(x);
plot(x,y1,x,y2);
xlabel('unghi [rad]');
ylabel('SIN & COS');
title('Finctii trigonometrice');
legend({'SINUS','COSINUS'},'Location','northeastoutside');
grid on;
figure;
plot(x,y3,'r');
hold on;
plot(x,y4,'m');
hold off;
xlabel('unghi [rad]');
ylabel('TAN & COT');
title('Finctii trigonometrice');
legend({'TANGENTA','COTANGENTA'},'Location','south');
grid on;
```





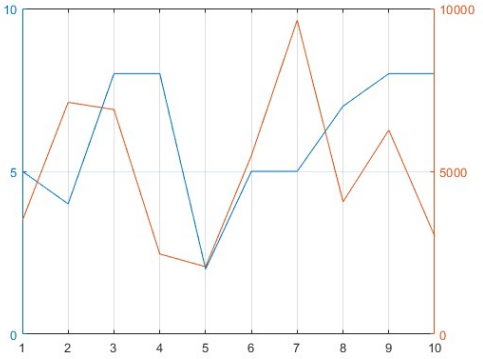
a) grafic sinus + cosinus b) grafic tangentă + cotangentă

Fig. 2.10. Grafice funcţii trigonometrice

Dacă există o diferență mare între valorile de pe ordonată, pentru două curbe suprapuse pe același grafic, atunci va fi necesar existența a două scări de reprezentare pe ordonată. Acest lucru se poate realiza cu funcția **plotyy**.

În Fig. 2.11 este prezentat programul şi reprezentarea grafică a datelor utilizând fincția **plotyy.**

```
x=1:10;
y1=randi([1,10],1,10);
y2=randi([1000,10000],1,10);
plotyy(x,y1,x,y2);
grid on;
```



a) linii de comandă b) grafic

Fig. 2.11. Grafic cu două ordonate

Se observă existența a două scări. Una între 0 şi 10, iar cealaltă între 0 şi 10000. Dacă am fi utilizat funcția plot atunci pe curba albastra nu s-ar fi observat variațiile, curba fiind aproximată cu o dreaptă foarte apropiată de abscisă.

Toate reprezentările grafice prezentate până în acest moment au utilizat scara lineară standard atât pentru abscisă cât şi pentru ordonată. Dacă există variații mari între valori atunci scara lineară nu este utilă. De exemplu pentru intesitatea sunetului sau luminii se utilizează scara logaritmică.

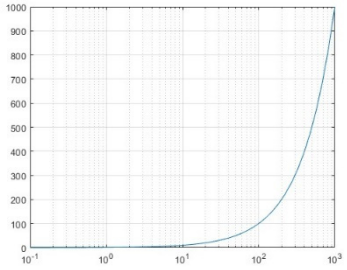
Dacă este necesară reprezentarea uneia sau ambelor axe la scară logaritmică în baza 10, acest lucru este posibil în Matlab prin utilizarea funcțiilor:

- **semilogx** – pentru reprezentarea abscisei la scară logaritmică;
- **semilogy** – pentru reprezentarea ordonatei la scară logaritmică;
- **loglog** – pentru reprezentarea ambelor axe la scară logaritmică;

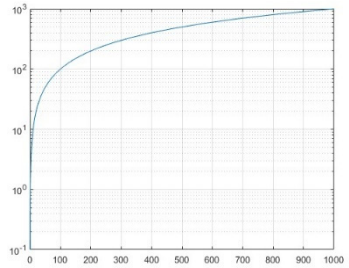
Funcțiile enumerate anterior sunt utilzate în următorul script (Fig. 2.12). Se generează un vector **x** care are 50 de elemente spațiate logaritmic între 10^{-1} și 10^{3}. Acest vector va fi reprezentat funție de el însuși în cele trei cazuri prezentate.

```
x=logspace(-1,3,50);
figure;
semilogx(x,x);
grid on;
figure;
semilogy(x,x);
grid on;
figure;
loglog(x,x);
grid on;
```

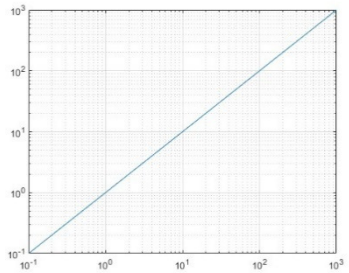
a) linii de comndă



b) scară logaritmică pe abscisă



c) scară logritmică pe ordonată



d) scară logaritmică abscisă și ordonată

Fig. 2.12. Reprezentare grafică la scară logaritmică în baza 10

Dacă se dorește afișarea grafică doar a punctelor (fără a fi unite printr-o curbă), se poate realiza utilizând proprietățile de formatare a funcției plot, sau utilizând funcția **scatter** (Fig. 2.13).

```
x=[1:10];
y=[28,37,41,58,60,63,66,83,89,95];
scatter(x,y);
grid on;
```

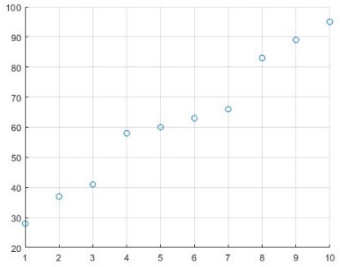


Fig. 2.13. Grafic utilizând funcția scatter

Dacă se dorește umplerea ariei de sub curba reprezentată grafic, se va folosi funcția **area** (Fig. 2.14)**.**

```
x=[1:10];
y=[28,37,41,58,60,63,66,83,89,95];
area(x,y);
grid on;
```

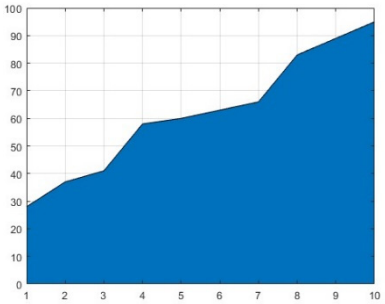


Fig. 2.14. Grafic utilizand funcția area

Toate funcțiile pentru reprezentări grafice prezentate sunt utilizate pentru grafice 2D. Dacă se dorește reprezentare grafică 3D, atunci se vor folosi următoarele funcții:

- **plot3** – reprezentare 3D a unei curbe;
- **mesh** – reprezentare 3D a discretizării unei suprafețe;
- **surf** – reprezentrea 3D a unei suprafețe.

La aceste reprezentări tridimensionale, toate proprietățile de formatare și design prezentate la funcțiile de reprezentare grafică 2D sunt valabile și în aceste cazuri. Singura diferență (obligativitate) sunt valorile de pe cea de a treia axa (axa Z).

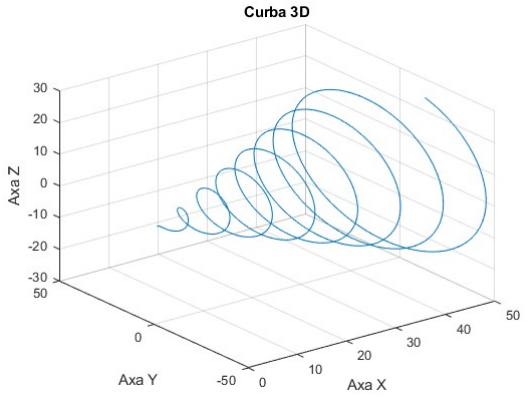
Pentru exemplificare se va crea un script care să genereze o curbă spațială, o discretizare (mesh) respectiv o suprafață (Fig. 2.15).

Pentru utilizarea funcției **plot3** avem nevoie de 3 vectori egali ca lungime, care să conțină coordonatele punctelor pe cele trei axe. Pentru reprezentarea unei discretizări sau suprafețe este necesar ca cele trei variabile (x,y,z) să fie sub formă matricială. Dacă coordonatele dintr-un plan sunt stocate sub formă de vector, generarea tuturor combinațiilor posibile poate fi realizată prin funcția **meshgrid**.

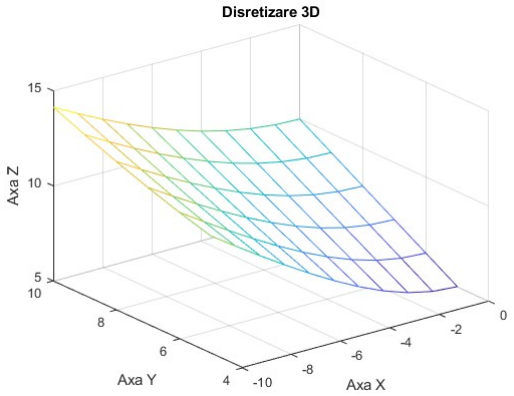
Pentru exemplificare generam un vector x=[1,2,3,4,5] și un vector y=[6,7,8,9,10]. Utilizând funcția meshgrid v-om genera matricele x1 şi x2 care vor avea 5 linii și 5 coloane.

Matricea x1 va avea 5 linii (atâtea elemente are vectorul y) și fiecare linie va conține elementele vectorului x (deci matricea va avea 5 coloane). Matricea y1 se va genera identic, cu mențiunea ca elementele vectorului y se vor plasa pe coloane. Pentru a reprezenta grafic un plan paralel cu planul XOY la o distantă egala cu 1, se va genera matricea z1 care trebuie să aibă aceeași dimensiune cu matricile x1 și y1, și a cărei elemente vor fi toate egale cu 1. În Fig. 2.16 este prezentat scriptul și rezultatul rulării acestuia.

```
x=1:0.1:50;
y=x.*sin(x);
z=x/2.*cos(x);
figure;
plot3(x,y,z);
grid on;
title('Curba 3D')
xlabel('Axa X');
ylabel('Axa Y');
zlabel('Axa Z');
[x1,y1] = meshgrid(-
10:0,5:10);
z1 = sqrt(x1.^2 + y1.^2);
figure;
mesh(x1,y1,z1);
title('Disretizare 3D')
xlabel('Axa X');
ylabel('Axa Y');
zlabel('Axa Z');
grid on;
figure;
surf(x1,y1,z1);
title('Suprafata 3D')
xlabel('Axa X');
```



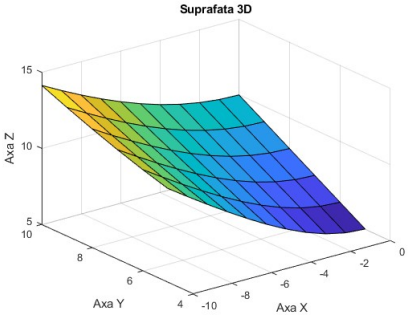
b) grafic curba 3D



c) discretizare 3D

```
ylabel('Axa Y');
zlabel('Axa Z');
grid on;
```

a) linii de comandă



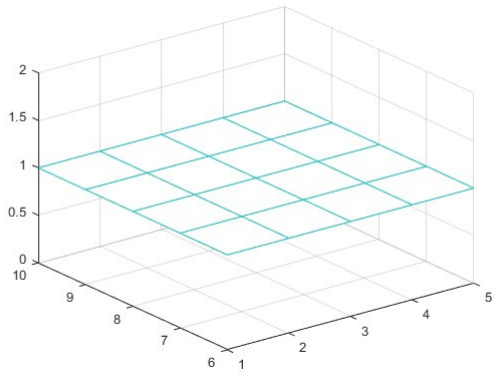
d) suprafață 3D

Fig. 2.15. Reprezentări grafice 3D

```
x=1:5;
y=6:10;
[x1,y1]=meshgrid(x,y);
z1=ones(size(x1));
mesh(x1,y1,z1);
```

```
x1 =
    1    2    3    4    5
    1    2    3    4    5
    1    2    3    4    5
    1    2    3    4    5
    1    2    3    4    5
y1 =
    6    6    6    6    6
    7    7    7    7    7
    8    8    8    8    8
    9    9    9    9    9
   10   10   10   10   10
z1 =
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
```

a) linii de comndă



b) grafic

Fig 2.16. Plan paralel cu XOY la distanța egală cu 1

De asemenea mai există posibilitatea reprezentării grafice a rezultatelor și sub alte forme:

- bare - (bar, bar3);
- sector de cerc - (pie, pie3);
- polar – (polar);
- histogramă – (hist);
- pareto – (pareto);

Pentru reprezentarea cantitativă a datelor se pot utiliza funcțiile **bar** și **pie**. Fiecare dintre aceste funcții având și varianta 3D. O reprezentarea polară se poate reprezenta cu ajutorul funcției **polar**. Pentru afisarea histogramelor se utilizează funcția **hist**. O altă soluție de afișare a frecvenței evenimentelor este diagrama Pareto care poate fi reprezentată cu ajutorul funcției **pareto.**

2.5 Citirea și scrierea datelor din fisiere externe

Există mai multe posibilități de introducere ale datelor inițiale necesare implementării oricărui algoritm, și anume:

- Definirea acestora în cadrul programului. Acest lucru este util dacă definim constante sau parametri care nu se modifică la rulările ulterioare ale programului;
- Posibilitatea introducerii de la tastatura de către utilizator. Acest lucru realizându-se cu ajutorul funcției **input**;
- Citirea datelor din fișiere externe. Acest lucru este util dacă pachetul de date de intrare este mare sau daca programul prelucrează date care se tot modifică. Pentru importul acestora se pot utiliza următoarele funcții: **fscanf, readtable, csvread, xlsread, load**.

Declararea directă a variabilelor în cadrul programului s-a prezentat când s-a vorbit despre variabile și se realizează prin atribuirea unei valori utilizând simbolul “= „ pentru atribuire. În continuare se va exemplifica utilizarea funcțiilor pentru citirea datelor din fișiere externe.

```
 % citiea din fisier text utilizand functia fscan
fisier=fopen('date.txt','r'); % deschidem pentru citire fisierul date.txt
data1=fscanf(fisier,'%f');, % citim datele numerice in baza 10 din fisier
fclose(fisier); % inchidem fisierul
% citirea datelor utilizand functia readtable
```

```
data2=readtable('date.txt');
% citirea datelor utilizand functia csvread
data3=csvread('date.dat');
% citirea datelor utilizand functia xlsread
data4=xlsread('date.xlsx');
% citirea datelor utilizand functia load
data5=load('date.mat');
```

Pentru citirea datelor din fişier text trebuie prima oară deschis pentru citire fişierul (**fopen**), apoi se vor citi date (**fscanf**) urmând să închidem fişierul (**fclose**). Pentru restul funcţiilor de citire a datelor este suficient să menţionam doar numele fişierului. Pentru funcţiile care citesc din fişiere de tip tabel avem posibilitatea de a defini doar o zonă care să fie citită.

O funcţie care este utilă pentru lucrul cu fişiere este **uigetfile**. Acestă funcţie ne oferă posibilitatea selectării unui anumit fişier prin navigarea într-o fereastra de selecţie.

De multe ori afişarea rezultatelor calculelor în fereastra de comenzi sau sub formă grafică nu este suficient, astfel se doreşte ca rezultatele să fie stocate sub formă tabelara. Programul Matlab permite exportul datelor în diferite fişiere şi sub diferite formate. Fiecărei funcţii de citire a datelor din fişier îi corespunde şi o funcţie care salvează datele în fişiere externe. Aceste funcţii sunt: **fprintf, writetable, csvwrite, xlswrite, save**.

Fişierele cele mai frecvent utilizate pentru stocarea datelor sub formă tabelară sunt fişierele de tip excel. Se vor detalia în continuare funcţiile de citire şi scriere în şi din acest tip de fisiere.

[numere,text,data]=xlsread(nume_fisier,foaie,zona);

Funcţia de citire a datelor din fişier excel este **xlsread**. Parametrii de intrare a funcţiei sunt:

- **nume_fisier** – care reprezintă numele fişierului care conţine datele care se vor citi. Dacă fişierul se află în directorul curent este suficient ca variabila **nume_fisier** să conţină doar numele acestuia, dar dacă fişierul este stocat într-o altă parte este necesară şi prezenţa căi de acces până la fişier;
- **foaie** – acestă variabilă poate fi numerică reprezentând din a câta filă se va face citirea (filele sunt numerotate de la 1 în ordinea din fişierul excel), sau poate fi alfanumerică reprezentând numele filei. Implicit variabila foaie este setată pe unu;

- **zona** – este o variabila alfanumerică care conține aria (porțiunea) care se dorește citită. Dacă nu este menționată zona atunci se citește conținutul întregii file.

Parametrii de ieșire a funcției sunt:

- **numere** – reprezintă o matrice care conține doar valorile numerice citite;
- **text** – reprezintă un tabel de celule care conține doar câmpurile alfanumerice citite;
- **data** – reprezintă un tabel de celule dar care conține toate datele citite.

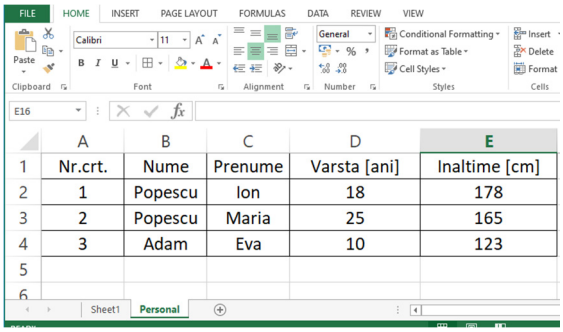
Pentru exemplificare s-a generat un fișier excel numit Date_Personal.xlsx (Fig. 2.17).

Dacă se dorește citirea tuturor datelor din foaia denumită **Personal,** acest lucru se poate realiza prin una din următoarele comenzi, unde pe langă numele fișierului se menționează fie numele fie numărul foii unde se găsesc datele dorite.

[numere,text,data]=xlsread('Date_Personal','Personal');

sau

[numere,text,data]=xlsread('Date_Personal',2);



	A	B	C	D	E
1	Nr.crt.	Nume	Prenume	Varsta [ani]	Inaltime [cm]
2	1	Popescu	Ion	18	178
3	2	Popescu	Maria	25	165
4	3	Adam	Eva	10	123
5					
6					

Fig. 2.17. Fișier excel inițial

După citirea datelor avem următoarele variabile:

```
numere =
    1   NaN   NaN    18   178
    2   NaN   NaN    25   165
    3   NaN   NaN    10   123

text =
  4×5 cell array
    {'Nr.crt.'}    {'Nume'   }    {'Prenume'}    {'Varsta [ani]'}    {'Inaltime [cm]'}
```

```
    {0×0 char }    {'Popescu'}    {'Ion'    }    {0×0 char      }    {0×0 char      }
    {0×0 char }    {'Popescu'}    {'Maria'  }    {0×0 char      }    {0×0 char      }
    {0×0 char }    {'Adam'   }    {'Eva'    }    {0×0 char      }    {0×0 char      }

data =
  4×5 cell array
    {'Nr.crt.'}    {'Nume'   }    {'Prenume'}    {'Varsta [ani]'}    {'Inaltime [cm]'}
    {[      1]}    {'Popescu'}    {'Ion'    }    {[         18]}    {[         178]}
    {[      2]}    {'Popescu'}    {'Maria'  }    {[         25]}    {[         165]}
    {[      3]}    {'Adam'   }    {'Eva'    }    {[         10]}    {[         123]}
```

Se observă că variabila **numere,** fiind o matrice, în celulele care nu conţineau valori numerice s-a completat **NaN** (not a number), iar variabila **text** care este de tip tablou de celule, conţine doar valorile alfanumerice, celule numerice fiind lăsate libere. Variabila **data** este de asemenea de tip tablou de celule, fiecare celulă fiind formatată conform conţinutului.

Pentru salvarea datelor în fişier excel se utilizează funcţia **xlswrite**.

xlswrite(fisier,variabila,foaie,celula)

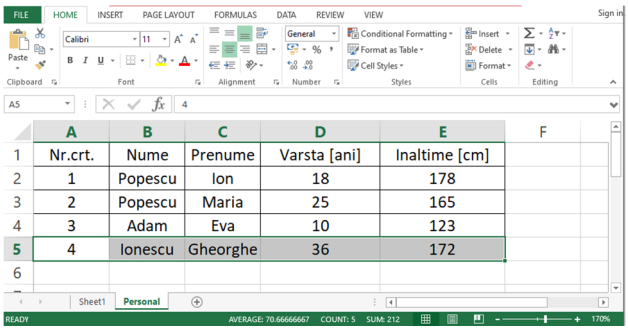
Acesta funcţie are doar parametrii de intrare:

- **fisier** – numele fişierului excel în care se doreşte salvarea datelor. Şi în acest caz dacă fisierul se doreşte salvat în altă locaţie decât cea curentă, trebuie menţionată calea. Dacă fisieul nu există, atunci el se va crea, iar dacă fişierul există, se vor completa datele în acesta;
- **variabila** – este numele variabilei care conţine valorile ce trebuie scrise în fisier;
- **foaie** – este numele filei unde se doreşte salvarea.
- **celula** – este numele celulei de la care în dreapta şi în jos se completează datele salvate.

Pentru exemplificare se va completa încă un rând în fişierul excel citit în exemplul anterior. Variabila **scriere** conţine valorile ce trebuie completate pe rândul al 5-lea din fila a doua a fişierului excel denumit **Date_Personal.xlsx**.

```
scriere={4,'Ionescu','Gheorghe',36,172};
xlswrite('Date_Personal.xlsx',scriere,2,'A5');
```

Rezultatul execuţiei acestor linii de comandă este prezentată în Fig. 2.18.



	A	B	C	D	E	F
1	Nr.crt.	Nume	Prenume	Varsta [ani]	Inaltime [cm]	
2	1	Popescu	Ion	18	178	
3	2	Popescu	Maria	25	165	
4	3	Adam	Eva	10	123	
5	4	Ionescu	Gheorghe	36	172	
6						

Fig. 2.18. Fişier excel final – după scriere

Fişierele de tip *.mat sunt fişiere dedicate programul Matlab pentru stocarea informaţiilor. Aceste fişiere pot fi scrise sau citite cu ajutorul funcţiilor **load** şi **save**. Aceste fişiere pot conţine oricâte variabile şi de orice tip. La orice moment fisierul poate fi modificat prin adaugara de variabile (append) respectiv rescrierea acestuia (overwrite). Din aceste tip de fişiere se pot citi toate variabilele odată sau prin precizarea numelor acestora, doar cele necesare.

2.6 Crearea funcţiilor în Matlab

Dacă o bucată de cod se repetă de mai multe ori, sau dacă anumite calcule trebuie efectuate în mai multe locaţii ale algoritmului atunci cea mai eficientă metodă este aceea de a crea o funcţie (subrutină) care să fie apelată ori de câte ori este nevoie.

```
function [iesire1, iesire2,…] = nume_functie(intrare1,intrare2,…)
bloc instrucţiuni;
end
```

Pentru a crea o astfel de subrutină, blocul de instrucţiuni trebuie cuprins între cuvintele cheie **function** şi **end**. Primul rând al funcţiei trebuie să mai conţină pe lângă cuvântul cheie **function** şi parametrii de ieşire a funcţiei, menţionaţi între paranteze pătrate, numele funcţiei (nume_funcţie prin care se va apela funcţia ori de cate ori este cazul) şi parametri de intrare ai funcţiei.

Blocul de instrucţiuni cuprinde o înşiruire de comenzi care pe baza parametrilor de intrare calculează (determină) parametri de ieşire ai funcţiei. Dacă funcţia creată se va utiliza doar în cadrul aplicaţiei curente atunci aceasta poate fi integrată la începutul scriptului. Dar dacă se doreşte ca funcţia să fie apelată şi din cadrul altor aplicaţii atunci este obligatoriu ca aceasta să fie salvată într-un script independent al cărui nume trebuie să fie identic cu cel al funcţiei.

În exemplul următor se crea o funcție denumită **mate**, și care are ca parametri de intrare variabilele numerice **a** și **b**, iar ca parametri de ieșire variabilele numerice **s** și **p** care reprezintă suma respectiv produsul parametrilor de intrare.

```
function [s,p] = mate(a,b)
   s=a+b;
   p=a*b;
end
```

La apelarea funcției prin comanda:

[suma,produsul]=mate(2,5);

rezultă **suma = 7** și **produsul = 10**.

Funcțiile pot fi apelate ori de câte ori este nevoie. Ce trebuie avut în vedere este ordinea definiri parametrilor de intrare și de iesire. Se observă că parametri de ieșire ai funcției sunt **s** și **p** dar în programul principal variabilei **suma** îi va fi atribuită valoarea variabilei **s**, iar variabilei **produsul** îi va fi atribuită valoarea variabilei **p**.

2.7 Baze de date

Pentru stocarea, organizarea și centralizarea datelor cea mai bună soluție este utilizarea unei baze de date. Limbajul Matlab permite accesarea bazelor de date prin intermediul modulului **Database Toolbox**. Acest modul facilitează accesul la baza de date, permițând adăugarea, inserarea, ștergerea, selectarea sau modificarea informațiilor din baza de date.

Bazele de date **Mysql** sunt cel mai des utilizate. Ele prezintă o securitate ridicată iar limbajul de gestionare a acestora este foarte simplu. Programul Matlab prin modulul „bază de date”, pune la dispoziția utilizatorului următoarele comenzi generale pentru gestiunea bazelor de date:

- **database –** realizează conexiunea la baza de date;
- **fetch –** caută informați în baza de date după anumite criteri;
- **exec –** permite executarea oricărei comenzi *sql* de gestionare a bazei de date.

Sintaxă:

conectare = database (db_name, utilizator, parola, driver, db_url);

Parametri de intrare ai funcției de conectare la baza de date sunt:

- **db_name** – numele bazei de date;
- **utilizator** – numele utilizatorului (username) de conectare la baza de date;
- **parola** – parola utilizatorului;
- **driver** – cu ajutorul cărui driver (jdbc, odbc, mysql, postgresql) se realizează conectarea la baza de date;
- **db_url** – adresa (cale + nume) bazei de date.

După executarea acestei comenzi variabila **conectare** va conține toate informațiile necesare conectări la baza de date. La fiecare interacțiune cu baza de date, este necesară utilizarea acestei variabile pentru a ne fi permis accesul la aceasta.

Pentru a căuta anumite informați în baza de date se utilizează comanda **fetch**.

Sintaxă:

rezultat = fetch (conectare, comanda_sql);

Parametri de intrare ai funcției **fetch** sunt: **conectare** (variabila rezultată în urma executări comenzi database) și **comanda_sql** care reprezintă instrucțiunea în limbajul *sql* pentru stabilirea criteriilor de căutare. În urma execuției acestei comenzi, variabila **rezultat** va conține informația căutată în baza de date pe baza instrucțiuni *sql* din variabila **comanda_sql**. Această comandă **fetch** permite doar căutarea informației în baza de date. Dacă se dorește modificarea, ștergerea, înserarea, adăugarea de informații în baza de date, acest lucru se realizează utilizând funcția **exec**.

Sintaxă:

exec (conectare, comanda_sql);

Funcția **exec** are ca parametri de intrare variabila **conectare** (legătura cu baza de date) și variabila **comanda_sql**, care reprezintă o instrucțiune de gestiune, a bazei de date, în limbajul *sql.*

Pentru a putea interacționa cu baza de date, pe lângă aceste funcți ale limbajului Matlab, mai este nevoie de cunoașterea instrucțiunilor de baza în limbajul *sql* [3], și anume:

- select – caută informați în baza de date;
- update – modifică informațiile din baza de date;
- insert into – adaugă informați în baza de date;
- delete – șterge informați din baza de date.

În continuare se vor prezenta sintaxele comenzilor *sql* pentru gestionarea bazelor de date din MySql.

SELECT camp1, camp2,….. FROM tabel [WHERE conditi];

Funcția **select** permite selectarea valorilor din coloanele denumite **camp1, camp2, etc** ale tabelului denumit **tabel.** Dacă se dorește selectarea tuturor coloanelor tabelului atunci se poate utiliza simbolul * în loc să menționăm numele tuturor coloanelor. De asemenea dacă se dorește selectarea doar a anumitor rânduri atunci prin utilizarea cuvântului cheie **where** se pot menționa condiți suplimentare de selecție.

UPDATE tabel SET camp1 = val1, camp2 = val2,….[WHERE conditi];

Instrucțiunea **update** este utilizată pentru a modifica valoarea **camp1** cu **val1**, **camp2** cu **val2**, etc a tabelului denumit **tabel**, dacă sunt îndeplinite condițiile impuse cu ajutorul cuvântului cheie **where**.

INSERT INTO tabel (camp1, camp2,….) VALUES (val1, val2,….);

Comanda **insert into** este utilizată pentru a introduce înregistrări noi în tabelul denumit **tabel**. **camp1, camp2, etc** reprezintă numele coloanelor unde se vor introduce valorile **val1, val2, etc**. Dacă se introduc valori pentru toate coloanele tabelului, atunci nu este obligatoriu să menționam numele acestora în cadrul comenzii, dar ordinea valorilor **val1, val2, etc** trebuie să coincidă cu ordinea coloanelor în cadrul tabelului.

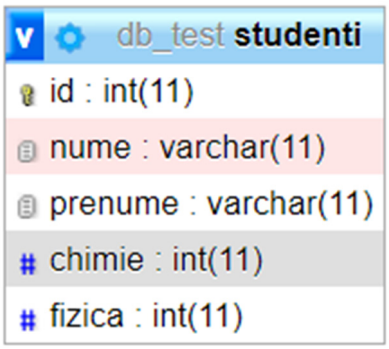
DELETE FROM tabel [WHERE conditii];

Funcția **delete** permite ștergerea condiționată a informațiilor din tabelul denumit tabel.

În continuare pentru exemplificare s-a creat o baza de date numită **db_test**, care conține un singur tabel denumit **studenti**. Tabelul conține 5 campuri:

- id – câmp numeric care conține id-ul;
- nume – câmp alfanumeric care conține numele studentului;
- prenume – câmp alfanumeric care conține prenumele studentului;
- chimie – câmp numeric care conține nota la chimie;
- fizica – câmp numeric care conține nota la fizică;

Inițial în baza de date sunt înregistrați trei studenți. În Fig. 2.19 se prezită structura şi conținutul tabelului denumit **studenti**, ai bazei de date din MySql.



a) structura

id	nume	prenume	chimie	fizica
1	Popescu	Ion	10	10
2	Ionescu	Vasile	7	8
3	Georgescu	Maria	9	10

b) conținutul

Fig. 2.19. Tabelul studenți al bazei de date

La rularea următoarelor linii de comandă se vor citii datele din baza de date, apoi se va însera, modifica și șterge anumite informații în baza de date.

```
% Gestionarea bazelor de date
% Curatarea cw si ws
clear;
clc;
% Citirea de la tastatura a datelor necesare conectării la db
% Se introduce numele utilizatorului
user=input('Introduceti numele utilizator = ','s');
% Se introduce parola
pass=input('Introduceti parola = ','s');
% Se introduce numele bazei de date
db=input('Introduceti numele bazei de date = ','s');
% Se defineste driver-ul utilizat
driver='com.mysql.jdbc.Driver';
% Se concateneaza locatia (calea) cu numele bazei de date
url=strcat('jdbc:mysql://localhost:3306/',db);
% Se realizeaza conectarea la baza de date
conectare=database(db,user,pass,driver,url);
% Comanda sql pentru selectarea tuturor datelor din tabel
query='select * from studenti';
% rezultat1 va contine toate datele din tabel
rezultat1=fetch(conectare,query)
% Comanda sql pentru selectarea inregistrarii cu id=2 din tabel
query='select * from studenti where id=2';
% rezultat2 va contine toate datele din tabel pentru inregistrarea cu id=2
rezultat2=fetch(conectare,query)
```

```
% Comanda sql pentru selectarea inregistrarii cu prenume=Vasile din tabel
query='select * from studenti where prenume="Vasile"';
% rezultat2 va contine toate datele din tabel pentru inregistratiele cu
% prenume=Vasile
rezultat3=fetch(conectare,query)
% Comanda sql de modificare a notei la chimie pentru studentul Ionescu
% Vasile
query='update studenti set chimie=9 where nume="Ionescu" and
prenume="Vasile"';
% se va executa comanda de modificare
exec(conectare,query);
% Verificam daca datele au fost modificate in tabel
query='select * from studenti';
% rezultat1 va contine toate datele din tabel
rezultat1=fetch(conectare,query)
% Comanda sql de adaugare a studentului Creanga Ion care are 6 la chimie si
% 8 la fizica
query='insert into studenti (nume,prenume,chimie,fizica) values
("Creanga","Ion",6,8)';
% se va executa comanda de adaugare
exec(conectare,query);
% Verificam daca a fost adaugata inregistrarea in tabel
query='select * from studenti';
% rezultat1 va contine toate datele din tabel
rezultat1=fetch(conectare,query)
% Comanda sql de stergere a studentului Ionescu Vasiele
query='delete from studenti where nume="Ionescu" and prenume="Vasile"';
% se va executa comanda de adaugare
exec(conectare,query);
% Verificam daca inregistrarea a fost stearsa
query='select * from studenti';
% rezultat1 va contine toate datele din tabel
rezultat1=fetch(conectare,query)
```

În fereastra de comenzi se cere ca utilizatorul să introducă userul, parola și numele bazei de date.

Introduceti numele utilizator = test

Introduceti parola = passtest

Introduceti numele bazei de date = db_test

Apoi se va afişa variabila **rezultat1** care conține toate informațiile din tabelul **studenţi** ai bazei de date.

```
rezultat1 =
  3×5 table
    id       nume          prenume      chimie    fizica
    __    _____________    _________    ______    ______
    1     {'Popescu'  }    {'Ion'   }      10        10
    2     {'Ionescu'  }    {'Vasile'}       7         8
    3     {'Georgescu'}    {'Maria' }       9        10
```

În variabila **rezultat2** se vor afla doar informațiile pentru înregistrarea care are câmpul **id** egal cu 2.

```
rezultat2 =
  1×5 table
    id       nume          prenume      chimie    fizica
    __    _____________    _________    ______    ______
    2     {'Ionescu'}      {'Vasile'}       7         8
```

```
rezultat3 =
  1×5 table
    id       nume          prenume      chimie    fizica
    __    _____________    _________    ______    ______
    2     {'Ionescu'}      {'Vasile'}       7         8
```

În variabile **rezultat3** se vor afla doar informațiile pentru înregistrarea care are câmpul **prenume** egal cu sirul de caractere ”Vasile”. Se observă că variabila **rezultat2** este identică cu **rezultat3** deoarece condițiile impuse fac referire la aceeași înregistrare.

Se observă în variabila **rezultat1** că nota la chimie pentru studentul ”Ionescu Vasile” a fost modificată din 7, ea devenind 9.

```
rezultat1 =
  3×5 table
    id       nume          prenume      chimie    fizica
    __    _____________    _________    ______    ______
    1     {'Popescu'  }    {'Ion'   }      10        10
    2     {'Ionescu'  }    {'Vasile'}       9         8
    3     {'Georgescu'}    {'Maria' }       9        10
```

Variabila **rezultat1** se observă că a fost din nou modificată prin inserarea în baza de date a studentului ”Creanga Ion” care are la chimie nota 6 și la fizică nota 8.

```
rezultat1 =
  4×5 table
    id       nume          prenume      chimie    fizica
    __    _____________    _________    ______    ______
    1     {'Popescu'  }    {'Ion'   }      10        10
```

```
    2    {'Ionescu' }    {'Vasile'}      9       8
    3    {'Georgescu'}    {'Maria' }      9      10
    4    {'Creanga' }    {'Ion'  }      6       8
```

Variabila **rezultat1** s-a mai midificat o dată prin stergerea din baza de date a studentului ”Ionescu Vasile”

```
rezultat1 =
  3×5 table
    id       nume          prenume     chimie    fizica
    __    _____________    _________    ______    ______

    1    {'Popescu' }    {'Ion' }      10        10
    3    {'Georgescu'}    {'Maria'}      9        10
    4    {'Creanga' }    {'Ion' }      6         8
```

În scripul prezentat se observă ca variabila **query** reprezintă un şir de caractere care conţine comanda *sql* pentru operaţia dorită. Acestă variabilă este transmisă spre baza de date prin intermediul instrucţiunii **fetch**, dacă se doreste citirea inregistrărilor din baza de date, respectiv prin instrucţiunea **exec**, dacă se transmite doar informaţii bazei de date fără a aştepta raspuns de la aceasta.

2.8 Interfeţe grafice

Programul Matlab pune la dispoziţia utilizatorilor un mod facil de creare a interfeţelor grafice utilizator. Există două module ce pot fi utilizate în acest scop, şi anume **Guide** respectiv **AppDesigner**. Modulul **Guide** nu va mai fi dezvoltat, mai mult chiar, în versiunile următoare fiind posibil sa nu mai fie disponibil. Modulul **AppDesigner** are mai multe opţiuni de dezvoltare grafică decât modulul **Guide** fiind mai uşor de utilizat de cei care nu au cunoştinţe temeinice de proiectare orientată pe obiecte. În continuare se vor prezenta noţiunile de bază pentru dezvoltarea unei interfeţe grafice în modulul **AppDesigner.**

Modului **AppDesigner** se poate fi apelat direct prin apasarea butonului **Desin App** din tabul **APPS** al meniului (Fig. 2.20), sau prin comanda appdesigner scrisă în ferestra de comenzi.

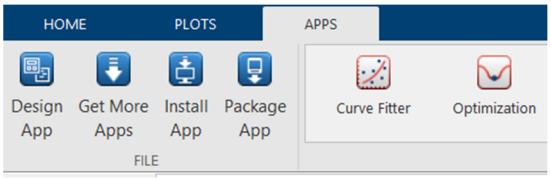


Fig. 2.20. Buton AppDesigner

Se va deschide fereastra modulului AppDesigner unde v-om putea crea interfața grafică pentru aplicația noastră. Această fereastra (Fig. 2.21) conține trei zone principale:

- **biblioteca de obiecte** – conține toate tipurile de obiecte disponibile. De aici prin tragere și plasare (drag & drop) se vor poziționa toate obiectele dorite în interfață;
- **interfață** – permite vizualizarea și aranjarea obiectelor în interfață atât in mod grafic cât și în mod text (cod sursă din spatele interfeței grafice) ;
- **obiectele utilizare în interfață** – sunt evidențiate obiectele utilizate, putând modifica proprietățile fizice (dimensiuni, culori, caracteristici) ale acestora.

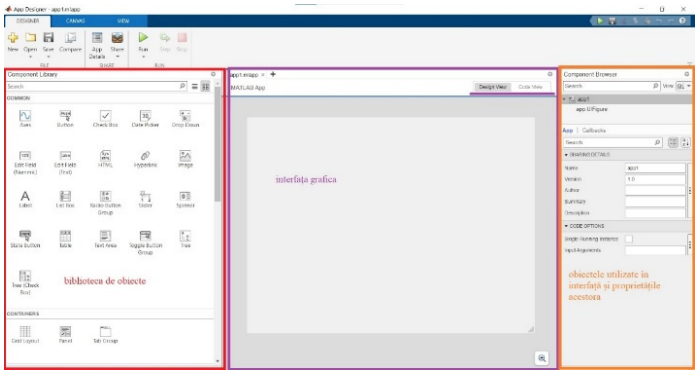


Fig. 2.21. Interfață AppDesigner

Toate obiectele din bibliotecă pot fi personalizate utilizând opțiunile disponibile ce apar în dreapta ecranului ca urmare a selectării unui obiect. Cele mai uzuale (utile) obiecte disponibile în bibliotecă sunt prezentate în tabelul 2.18.

Tabelul 2.18. Biblioteca de obiecte

A Label	utilizat pentru etichetat.
123 Edit Field (Numeric)	permite introducerea datelor numerice de către utilizator. conține o singură linie, nu permite caractere speciale.

 Edit Field (Text)	permite introducerea datelor alfanumerice de către utilizator. conține o singura linie.
 Text Area	definește o zonă unde se va afișa un text. Acesta poate fi modificat, și poate conține caractere speciale cum ar fi: tab, linie nouă, indentare, etc
 List Box	permite afișarea datelor sub formă de listă și selectarea unui element al listei
 Drop Down	permite afișarea datelor sub forma de listă ce se deschide la apăsare. Ca funcționalitate este identică cu lista (list box) doar vizual diferă.
 Button	Un buton cu o singură stare (buton push – sonerie)
 State Button	Un buton cu 2 stări (on-off – întrerupător)
 Toggle Button Group	Grup de butoane, la un moment dat doar o stare (buton) poate fi activă.
 Check Box	Variabila booleană (adevărat sau fals)
 Radio Button Group	Mai multe variabile booleene dar doar una poate fi adevărată la un moment dat.

	Afișarea datelor sub formă tabelară
	zonă în care se por afișa grafice, imagini, obiecte, etc
	permite vizualizarea imaginilor
	Bară culisantă. Va returna o valoare numerică cuprinsă în limitele definite.
	Contor identic cu bara culisantă doar grafic diferit. Acesta având și posibilitatea de introducere a unei valori de la tastatura, valoare cuprinsă între limitele definite.
	Panou, zonă care permite gruparea mai multor obiecte
	Grup de panouri (taburi) care permite gruparea obiectelor dar și posibilitatea de apelare a unei funcții la schimbarea selecției panourilor
	Un meniu care apare la click dreapta pe un obiect
	Bara de meniu. Permite crearea unui meniu personalizat
	Bara de butoane. Permite amplasarea de butoane tip apăsare sau cu reținere.

	Ceas indicator. Există mai multe modele grafice de ceasuri indicatoare. Valoarea numerică dorită va fi prezentată vizual pe indicator.
	Comutator cu mai multe stări. Există mai multe tipuri de comutatoare. Acestea prin selectare permite alegerea diferiților parametrii.
	Indicator boolean

Fiecare obiect care este utilizat în cadrul aplicației dezvoltate va avea un nume. Referirea în cadrul codului (programului) la oricare dintre obiectele utilizate se va realiza prin comnda:

app.nume_obiect.proprietate

Pentru accesarea sau modificarea proprietăților unui obiect se începe cu cuvântul cheie **app**, urmat de numele obiectului apoi de proprietatea dorita, separate de simbolul punct. Ținând cont că unui obiect i se pot atribui diferite valori precum și modifica caracteristicile sunt puse la dispoziție foarte multe proprietăți. În continuare se vor prezenta cele mai frecvent utilizate proprietăți ale obiectelor:

- Position – reprezintă poziția obiectului în interfața grafică;
- FontName – tipul de font utilizat;
- FontSize – dimensiunea fontului;
- FontColor – culoarea fontului;
- Visible – vizibilitatea obiectului poate fi setată **on** sau **off**
- Text – textul afișat de obiect. Acesta poate fi modificat oricand;
- Value – valoarea obiectului;
- Limits – setarea limitelor în cazul obiectelor tip slider, spinner, numeric editfield;
- Step - setarea incrementului utilizat în cazul obiectelor tip slider, spinner, numeric editfield;
- Editable – se permite sau nu modificare de către utilizator a datelor din câmpurile editabile;
- Items – reprezintă opțiunile pentru obiectele dropdown sau list (adică ce se afişează);

- ItemsData – reprezintă valoarea fiecărei opțiuni a obiectelor de tip dropdown sau lista. Dacă această proprietate este omisă atunci prin selectarea unei opțiuni rezultatul v-a fi textul selectat (Items – ul selectat).
- Data – valorile unui tabel;
- ColumnName – numele coloanelor unui tabel;
- ColumnWidth – dimensiunea coloanelor unui tabel;
- ColumnEditable – posibilitatea editării valorilor unui tabel de către utilizator;
- RowName – numele rândurilor unui tabel;
- Children – dacă afișăm mai multe grafice într-un obiect de tip UIAxis atunci fiecare grafic va putea fi formatat individul, deoarece pentru fiecare s-a creat cate o astfel de proprietate.

Pentru o simplă exemplificare a modului de utilizare a proprietăților obiectelor se va crea o interfață grafică utilizator cu trei obiecte. Un obiect tip etichetă (label), unul de tip listă cu cinci opțiuni și unul de tip indicator (Fig. 2.22).

Se observă că acestor obiecte le-a fost definit doar numele, formatarea obiectelor fiind cea de bază. Se dorește ca titlul afișat să fie „Acesta este titlul aplicatiei”, cu fontul TimesNewRoman de 24 pixeli, de culoare roșie. Opțiunile listei să fie: Optiune 1, Optiune 2, Optiune 3, Optiune 4, Optiune 5 (Items), iar valorile corespunzătoare fiecărei opțiuni să fie: 1,2,3,4,5 (ItemsData). Indicatorul să fie și el limitat între 1 și 5. Funcție de ce opțiune alegem din listă, valoarea acesteia va apărea pe indicator.

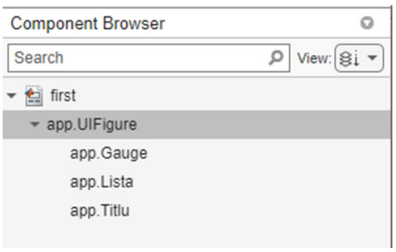


Fig. 2.22. Obiectele utizate în interfața grafică utilizator

Cele trei obiecte sunt denumite :

- Titlu – obiectul de tip eticheta;
- Lista – obiectul de tip listă;
- Gaudge – obiectul de tip indicator.

a) design intefață b) rulare interfață

Fig. 2.23. Interfața Grafică

În Fig. 2.23a se observă amplasarea obiectelor. Obiectele utilizate sunt formatate doar în momentul rulări aplicației. Personalizarea acestora relizânduse prin funcția ce rulează la pornirea aplicației (startup function). În Fig. 2.23b se observă formatarea obiectelor. Funcția de pornire are următoarea structură:

```
function startupFcn(app)
        % se defineste textul afisat
        app.Titlu.Text='Acesta este titlul aplicatiei';
        % se stabileste tipul fontului
        app.Titlu.FontName='TimesNewRoman';
        % se stabileste dimensiunea fontului
        app.Titlu.FontSize=24;
        % se stabileste culoarea fontului
        app.Titlu.FontColor='red';
        % se definesc optiunile afisate in lista
        app.Lista.Items={'Optiune 1','Optiune 2','Optiune 3','Optiune 4',…
        'Optiune 5'};
        % se stabilesc valorile pentru fiecare optiune
        app.Lista.ItemsData={1,2,3,4,5};
        % se limiteaza indicatorul
        app.Gauge.Limits=[1,5];
        % se formateaza scala indicatorului
        app.Gauge.MajorTicks=1:5;
end
```

De asemenea s-a creat și o funcție care la modificarea opțiunii din listă sa afișeze rezultatul în indicator.

```
function ListaValueChanged(app, event)
        app.Gauge.Value = app.Lista.Value;
end
```

Pentru a crea funcți se dă click dreapta pe obiectul pentru care se definește funcția apoi se selectează opțiunea **Callbacks**. Dacă obiectul respectiv nu are funcție atașată atunci va apărea opțiunea **Add Callbacks**, altfel creând o funcție pentru acel obiect. Daca obiectul are atașată o funcție atunci va apărea opțiunea de **Go To Callback** care ne duce direct la codul sursă al funcției acest lucru facilitând accesul la aceasta.

În Fig. 2.24 se observă că în codul sursă modulul **AppDesigner** crează funcțiile dorite de utilizator și permite modificarea acestora doar. Acest lucru benefic deoarece nu putem modifica din greșeală structura aplicației dezvoltate.

```
        % Code that executes after component creation
        function startupFcn(app)
            % se defineste textul afisat
            app.Titlu.Text='Acesta este titlul aplicatiei';
            % se stabileste tipul fontului
            app.Titlu.FontName='TimesNewRoman';
            % se stabileste dimensiunea fontului
            app.Titlu.FontSize=24;
            % se stabileste culoarea fontului
            app.Titlu.FontColor='red';
            % se definesc optiunile afisate in lista
            app.Lista.Items={'Optiune 1','Optiune 2','Optiune 3','Optiune 4
            % se stabilesc valorile pentru fiecare optiune
            app.Lista.ItemsData={1,2,3,4,5};
            % se limiteaza indicatorul
            app.Gauge.Limits=[1,5];
            % se formateaza scala indicatorului
            app.Gauge.MajorTicks=1:5;
        end

        % Value changed function: Lista
        function ListaValueChanged(app, event)
            app.Gauge.Value = app.Lista.Value;
        end
    end
```

Fig. 2.24. Cod sursă funție

De asemenea variabilele definite într-o funcție sunt definite ca variabile locale, adică accesibile doar în cadrul funcției în care au fost definite. Dacă dorim utilizarea unor variabile și în alte funcții ale aceeași aplicații, trebuie să le definim ca variabile globale sau ca proprietăți. Dacă definim variabilele ca proprietatea atunci referirea la acestea se face prin **app.nume_variabia** de oriunde din codul sursă. Dacă definim variabilele de tip global atunci ele pot avea orice nume (bineînțeles să înceapă cu un caracter alfanumeric și să nu conțină caractere speciale) și sunt accesibile în toate funcțiile în care sunt definite ca variabile globale, acest lucru realizându-se prin enumerarea numelor variabilelor după cuvântul cheie **global**.

Pentru exemplificarea variabilelor globale și locale se va crea o interfață grafică utilizator (Fig. 2.25), care să conțină două câmpuri editabile, un buton care prin apăsare va concatena textul introdus în cele două câmpuri și va afișa rezultatul într-un al treilea câmp dar care nu este editabil. Pentru a închide fereastra curentă (interfața grafică utilizator curentă) se va crea un buton în acest sens.

Fig. 2.25. Interfață exemplu concatenare

Cele două câmpuri editabile se vor numi **VariabilaAEditField** respectiv **VariabilaBEditField**. Butonul de concatenare se numește **ConcatenareButton**, iar câmpul unde se afișează concatenarea are denumirea **ABEditField**. Butonul de închidere este numit **XButton.**

Se va crea variabila Variabila_B ca o proprietate privată. Există si posibilitatea de creare ca proprietate publică, care față de cea privată este vizibilă pentru toate clasele de obiecte.

```
properties (Access = private)
    Variabila_B % variabila "Variabila_B" definita ca proprietate privata
End
```

În cadrul funcției de pornire se va seta non editabil doar câmpul unde se dorește afișarea concatenării. Implicit obiectele de tip EditField sunt editabile.

```
function startupFcn(app)
        app.ABEditField.Editable="off";
end
```

În funcția care rulează odată cu modificarea textului din câmpul **VariabilaAEditField** se va defini variabila globală **Variabila_A** care va conține șirul de caractere introdus în câmp.

```
function VariabilaAEditFieldValueChanged(app, event)
        global Variabila_A;
        Variabila_A = app.VariabilaAEditField.Value;
end
```

În funcția care rulează odată cu modificarea textului din câmpul **VariabilaBEditField** se va atribui variabilei (definită ca proprietate) **app.Variabila_B** șirul de caractere introdus în câmp.

```
function VariabilaBEditFieldValueChanged(app, event)
        app.Variabila_B = app.VariabilaBEditField.Value;
end
```

Prin apăsarea butonului de concatenare se va realiza concatenarea celor șiruri introduse și afișarea acestora în câmpul **ABEditField.** Concatenarea se realizează chiar dacă cele două variabile cu proprietăți diferite.

```
function ConcatenareButtonPushed(app, event)
        global Variabila_A;
        app.ABEditField.Value=strcat(Variabila_A,app.Variabila_B);
end
```

Pentru închiderea ferestrei curente se utilizează funcția **closereq()**.

```
function XButtonPushed(app, event)
        closereq();
end
```

3 APLICAȚII DEZVOLTATE ÎN MATLAB

Pentru a evidenția instrucțiunile și funcțiile prezentate în capitolul anterior se vor realiza câteva aplicații în Matlab pentru rezolvarea problemelor propuse. Problemele rezolvate în continuare nu sunt de o complexitate ridicată, dorindu-se mai mult aprofundarea cunoștințelor de bază studiate până în acest moment.

3.1 Aplicație 1 – Ecuația de gradul doi

Prima problemă propusă este rezolvarea ecuației de gradul doi. Ecuația de gradul doi are forma $a*x^2+b*x+c=0$. Se vor introduce de la tastatură coeficienții ecuației, după care se va afișa delta și soluțiile ecuației. Problema se va rezolva prin două moduri. În primul mod se va crearea unu script pentru rezolvarea ecuației. În cadrul scriptului se vor utiliza funcțiile **if** și **while** pentru a verifica condițiile și pentru a repeta calculul soluțiilor ecuației ori de câte ori dorește utilizatorul. În cel de al doilea mod se va crea o interfață grafică utilizator, iar pentru rezolvarea ecuației se va folosi funcția **roots** existentă în Matlab.

Modul 1 - script

```
% rezolvare ecuatie grad 2
% se curață command window si workspace-ul
clear;
clc;
% inițializăm condiție cu 1 pentru a fi adevarata conditia pentru
% functia while
conditie=1;
while conditie==1
   clc;
   % se citesc coeficientii ecuatiei de gradul 2
   a=input('introduceti a=');
   b=input('introduceti b=');
   c=input('introduceti c=');
   % se calculeaza delta
   delta=b^2-4*a*c;
   % se afiseaza in cw delta
   disp(horzcat('Delta = ',num2str(delta)));
```

```
   % se calculeaza si afiseaza solutiile functie de delta
   if delta==0
      x=(-b)/(2*a);
      disp(horzcat('Ecuatia are solutie unica x= ',num2str(x)));
   elseif delta>0
      disp('Ecuatia are solutii reale');
      x1=(-b-sqrt(delta))/(2*a);
      disp(horzcat('X1 = ',num2str(x1)));
      x2=(-b+sqrt(delta))/(2*a);
      disp(horzcat('X2 = ',num2str(x2)));
   else
      disp ('Ecuatia are solutii complexe');
      x1=(-b-sqrt(delta))/(2*a);
      disp(horzcat('X1 = ',num2str(x1)));
      x2=(-b+sqrt(delta))/(2*a);
      disp(horzcat('X2 = ',num2str(x2)));
   end
% se afiseaza in cw optiunile pentru reluarea calculului
   disp('Doriti sa reluati aplicatia?');
   disp('1 - Da');
   disp('2 - Nu');
   conditie=input('Introduceti optiunea dvs ');
% optiunea utilizatorului trebuie să fie 1 sau 2 altfel o reintroducem
   while (conditie~=1)&&(conditie~=2)
      disp('Optiunea dvs nu este permisa');
      conditie=input('Introduceti optiunea dvs ');
   end
end
```

Prin rularea acestui script în fereastra command window va aparea:

```
introduceti a=2
introduceti b=15
introduceti c=1
Delta = 217
Ecuatia are solutii reale
X1 = -7.4327
X2 = -0.06727
Doriti sa reluati aplicatia?
1 - Da
```

2 - Nu
Introduceti optiunea dvs

Modul 2 – interfață grafică

În fig. 3.1 este prezentată interfața grafică utilizator, realizată pentru rezolvarea ecuației de gradul doi. Aceasta conține trei câmpuri numerice editabile (a,b și c) pentru introducerea coeficienților ecuației, un câmp numeric needitabil (delta) pentru afișarea valorii delta, două câmpuri alfanumerice needitabile (x1 și x2) pentru afișarea soluțiilor ecuației, un obiect de tip **radio button** care afișează tipul soluțiilor ecuației, un buton de calcul și un buton de închidere. Câmpurile pentru afișarea soluției trebuie să fie de tip text deoarece în cazul soluțiilor complexe apare caracterul ”i” care nu poate fi afișat dacă câmpul este de tip numeric

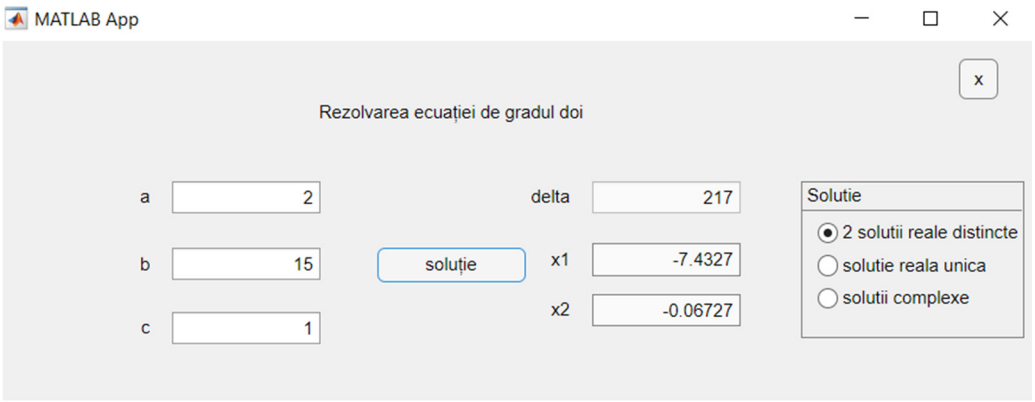


Fig. 3.1. Interfață grafică – Ecuaţia de gradul doi

Aplicția are definite două funcții. O functie atașată butonului de închidere care conține instrucțiunea **closereq(),** pentru închiderea interfeței și o funcție atașată butonului de calcul, care efectuază calculele necesare şi este prezentată în continuare.

```
function solutieButtonPushed(app, event)
    % comanda roots calculează soluțiile ecuației
    solutie=roots([app.aEditField.Value,app.bEditField.Value,…
    app.cEditField.Value]);
    % se calculeaza delta
    delta=app.bEditField.Value^2-…
    4*app.aEditField.Value*app.cEditField.Value;
    % se afiseaza rezultatul delta
```

```
        app.deltaEditField.Value=delta;
        % instructiunea if seteaza obiectul radio button
        % functie de valoare delta va fi selectata una din cele trei optiuni
        % posibile ale obiectului radio button
        if delta==0
                app.solutierealaunicaButton.Value=true;
        elseif delta>0
                app.solutiirealedistincteButton.Value=true;
        else
                app.solutiicomplexeButton.Value=true;
        end
        % se afiseaza solutiile
        app.x1EditField.Value=num2str(solutie(1));
        app.x2EditField.Value=num2str(solutie(2));
end
```

3.2 Aplicație 2 – Reprezentarea grafică a unei funcții matematice

Să se reprezinte grafic funcția y=3x^3+2x^2-5x+1, pe intervalul [Xmin, Xmax]. Limitele intervalului și numărul de puncte utilizat pentru reprezentarea grafică se introduce de la tastatură.

Pentru rezolvarea acestei probleme se va crea un script care va permite introducere de către utilizator al intervalului de afișare a funcției matematice precum și numărul de puncte în care să se calculeze funcția. Pe baza acestor parametrii se vor efectua calculele și se va reprezenta grafic funcția, graficul fiind generat într-o figură. Scriptul creat are următoarea structură:

```
% grafic functie y=3x^3+2x^2-5x+1
% se curate ws si cw
clear;
clc;
% se genereaza o figura pentru reprezentarea grafica
figure;
disp ('Introduceti intervalul [Xmin, Xmax]');
xmin=input('introduceti Xmin= ');
xmax=input('introduceti Xmax= ');
puncte=input('introduceti nr de puncte in care se va calcula functia');
% se genereaza vectorul ce contine valorile reprezentate pe abscisa
x=linspace(xmin,xmax,puncte);
% se calculeaza valoarea functiei y pentru fiecare valoare x
y=3*x.^3+2*x.^2-5*x+1;
```

```
% se realizeaza reprezentarea grafica
plot(x,y);
% se formateaza graficul
title('y=3x^3+2x^2-5x+1');
xlabel('axa X');
ylabel('axa Y');
grid on;
% se stabileste scar ape axa x si y
x1=min(x)-10;
x2=max(x)+10;
y1=min(y)-10;
y2=max(y)+10;
axis([x1 x2 y1 y2]);
```

Prin rularea scriptului prezentat în fereastra de comenzi apare:

Introduceti intervalul [Xmin, Xmax]
introduceti Xmin= **-50**
introduceti Xmax= **100**
introduceti nr de puncte in care se va calcula functia **50**

După introducerea parametrilor de intrare în figura creată va apărea graficul dorit (Fig. 3.2).

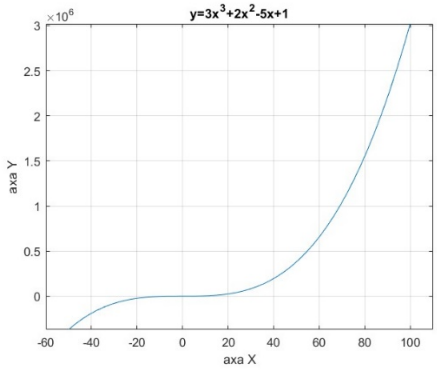


Fig. 3.2. Grafic funcție y=3x^3+2x^2-5x+1

3.3 Aplicație 3 – Modelul geometric direct al unui braț robotic

Calculul modelului geometric direct pentru un braț robotic cu șase grade de libertate utilizând convenția Denavit - Hartenberg.

În Fig. 3.3 se prezintă schema cinematică pentru bratul robotic RTTRRR. Brațul robotic este reprezentat ca un lant cinematic deschis, format din 6 cuple (4 cuple de rotație cu axă fixă și 2 de translație), ordinea și direcția acestora fiind prezentate în schema cinematică. De asemenea sunt notate cu l1, l2,…., l6,

lungimea fiecărui element al brațului robotic, iar varibilele cuplelor vor fi notate cu q1, q2,…., q6.

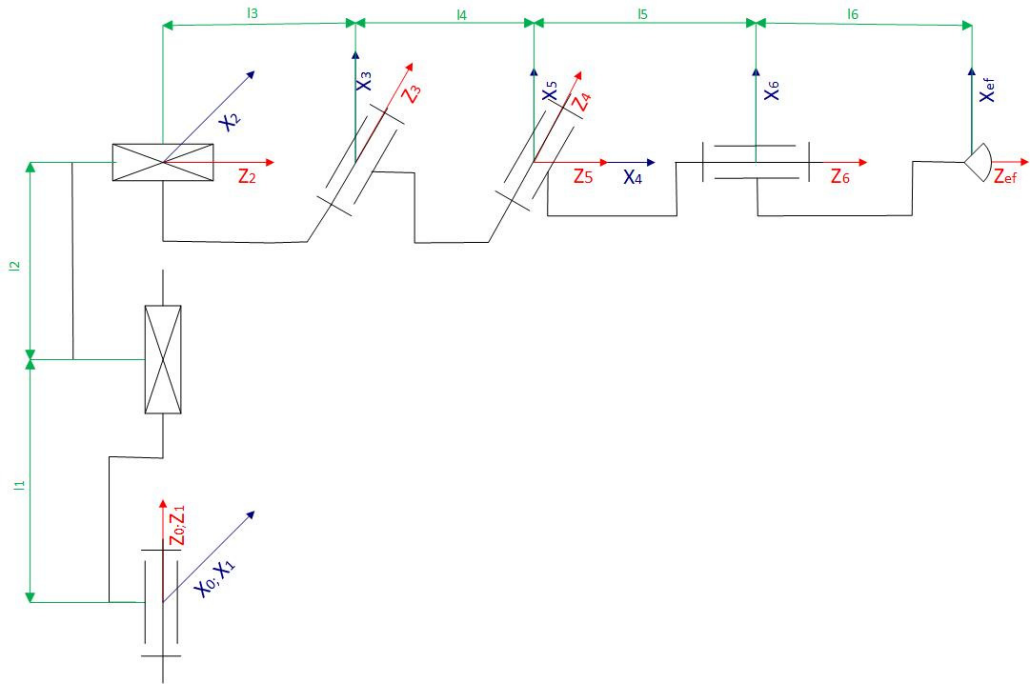


Fig. 3.3. Schema Cinematica a bratului robotic RTTRRR

Prin aplicarea convenției Denavit – Hartenberg se stabilesc parametrii geometrici prezentați în Tabelul 3.1. Pe baza acestor parametri se vor calcula matricele de transformare și cele generale. Modelul direct realizează legătura dintre coordonatele cuplelor și coordonatele operaționale.

Tabelul 3.1. Parametrii geometrici

Elementul	**L**	**α**	**d**	$\boldsymbol{\theta}$
1	0	0	0	q_1
2	0	$\frac{\pi}{2}$	$l_1 + l_2 + q_2$	0
3	0	$\frac{\pi}{2}$	$l_3 + q_3$	$\frac{\pi}{2}$
4	l_4	0	0	$q_4 + \frac{\pi}{2}$
5	0	$-\frac{\pi}{2}$	0	$q_5 - \frac{\pi}{2}$
6	0	0	l_5	q_6

Pentru calculul modelului geometric direct se va crea un program principal (script) și patru funcții pentru calculul matricelor de transformare, poziției inițiale, poziției finale respectiv a traiectoriei.

În cadrul programului principal se definesc parametri inițiali, se accesează funcțiile de calcul al matricelor de transfer, a poziției inițiale, a poziției finale, a traiectoriei și se afișează grafic rezultatele.

Programul Principal

```
% model geometric direct robot RTTRRR
% curatare
clear;
clc;
% declarere simboluri si constante
syms l1 l2 l3 l4 l5 l6 q1 q2 q3 q4 q5 q6;
fs=10;  % fs=factor scara pt scalarea axelor sistemelor de referinta
pas=10; % nr de pozitii in care se calculeaza traiectoria
% declarare parametrii geometrici
l=[0 0 0 l4 0 0];                    % lungimea L
a=[0 pi/2 pi/2 0 -pi/2 0];           % unghiul alfa
d=[0 l1+l2+q2 l3+q3 0 0 l5];         % distanta d
t=[q1 0 pi/2 q4+pi/2 q5-pi/2 q6];    % unghiul teta
% calculul matricelor de transfer
for i=1:6
T{i}=calcul_T(l(i),a(i),d(i),t(i)); % apelare functie calcul_T
end
% declaram valorile initiale
l1=20; %[cm]
l2=30; %[cm]
l3=10; %[cm]
l4=15; %[cm]
l5=20; %[cm]
l6=5;  %[cm]
q1=0;  %[rad]
q2=0;  %[rad]
q3=0;  %[rad]
q4=0;  %[rad]
q5=0;  %[rad]
q6=0;  %[rad]
valori_initiale=[l1,l2,l3,l4,l5,l6,q1,q2,q3,q4,q5,q6];
valori_finale=[l1,l2,l3,l4,l5,l6,q1,q2,q3,q4,q5,q6];
```

```
etichete={'Baza+C1' 'C2' 'C3' 'C4' 'C5' 'C6' 'EF'};
% afisarea pozitiei initiale
% apelare functie pozitia_initiala
[C,G,p,Cef]=pozitia_initiala(valori_initiale,T,fs);
plot3(C(:,1),C(:,2),C(:,3),'bx-');
grid on;
title('Pozitia initiala a robotului RTTRRR');
xlabel('Axa X');
ylabel('Axa Y');
zlabel('Axa Z');
xlim([-50 50]);
ylim([-50 50]);
zlim([0 65]);
text(C(:,1),C(:,2),C(:,3),etichete);
% calculul pozitiei robotului pentru o miscare individuala in fiecare cupla
% de rotatie cu pi/4 [rad] si cu 10 [cm] in cuplele de translatie
for i=1:6
figure;
if i==2||i==3
   valori_finale(i+6)=10;
   sir=' cu 10 [cm]';
else
   valori_finale(i+6)=pi/4;
   sir=' cu pi/4 [rad]';
end
M=pozitia_finala(valori_finale,G);
plot3(C(:,1),C(:,2),C(:,3),'bx-',M(:,1),M(:,2),M(:,3),'rx-');
grid on;
title(horzcat('Pozitia robotului RTTRRR cu miscare in cupla q',num2str(i),sir));
xlabel('Axa X');
ylabel('Axa Y');
zlabel('Axa Z');
xlim([-50 50]);
ylim([-50 50]);
zlim([0 65]);
text(C(:,1),C(:,2),C(:,3),etichete);
text(M(:,1),M(:,2),M(:,3),etichete);
valori_finale(i+6)=0;
end
traiectorie(l1,G,pas,fs,p,C,Cef,etichete)
```

Funcția de calcul a matricelor de transfer

```
function [T]=calcul_T(lungime,alfa,distanta,teta)
T=[cos(teta) -sin(teta)*cos(alfa) sin(teta)*sin(alfa) lungime*cos(teta);
   sin(teta) cos(teta)*cos(alfa) -cos(teta)*sin(alfa) lungime*sin(teta);
   0 sin(alfa) cos(alfa) distanta;
   0 0 0 1];
end
```

Funcția poziția inițială

```
function [C,G,p,Cef]=pozitia_initiala(v,T,fs)
l1=v(1);
l2=v(2);
l3=v(3);
l4=v(4);
l5=v(5);
l6=v(6);
q1=v(7);
q2=v(8);
q3=v(9);
q4=v(10);
q5=v(11);
q6=v(12);
% pozitia cuplei 1
C1=[0 0 0];
% pozitia cuplei 2
C2=[0 0 l1+q2];
% pozitia cuplei 3
G.G20=T{1}*T{2};
G20s=subs(G.G20);
C3=[G20s(1,4) G20s(2,4) G20s(3,4)];
% pozitia cuplei 4
G.G30=T{1}*T{2}*T{3};
G30s=subs(G.G30);
C4=[G30s(1,4) G30s(2,4) G30s(3,4)];
% pozitia cuplei 5
G.G50=T{1}*T{2}*T{3}*T{4}*T{5};
G50s=subs(G.G50);
C5=[G50s(1,4) G50s(2,4) G50s(3,4)];
% pozitia cuplei 6
G.G60=T{1}*T{2}*T{3}*T{4}*T{5}*T{6};
```

```
G60s=subs(G.G60);
C6=[G60s(1,4) G60s(2,4) G60s(3,4)];
% pozitia efectorului final
Tef=[1 0 0 0;
    0 1 0 0;
    0 0 1 l6;
    0 0 0 1];
G.Gef=T{1}*T{2}*T{3}*T{4}*T{5}*T{6}*Tef;
Gefs=subs(G.Gef);
Cef=[Gefs(1,4) Gefs(2,4) Gefs(3,4)];
% orientarea axelor pt pozitia initiala
ni=[Gefs(1,1) Gefs(2,1) Gefs(3,1)];
oi=[Gefs(1,2) Gefs(2,2) Gefs(3,2)];
ai=[Gefs(1,3) Gefs(2,3) Gefs(3,3)];
p.xi=Cef+ni*fs;
p.yi=Cef+oi*fs;
p.zi=Cef+ai*fs;
% matricea pozitiilor initiale a cuplelor
C=[C1;C2;C3;C4;C5;C6;Cef];
```

Funcția poziția finală

```
function [M]=pozitia_finala(v,G)
l1=v(1);
l2=v(2);
l3=v(3);
l4=v(4);
l5=v(5);
l6=v(6);
q1=v(7);
q2=v(8);
q3=v(9);
q4=v(10);
q5=v(11);
q6=v(12);
M1=[0 0 0];
M2=[0 0 l1+q2];
G20s=subs(G.G20);
M3=[G20s(1,4) G20s(2,4) G20s(3,4)];
G30s=subs(G.G30);
M4=[G30s(1,4) G30s(2,4) G30s(3,4)];
```

```
G50s=subs(G.G50);
M5=[G50s(1,4) G50s(2,4) G50s(3,4)];
G60s=subs(G.G60);
M6=[G60s(1,4) G60s(2,4) G60s(3,4)];
Gefs=subs(G.Gef);
Mef=[Gefs(1,4) Gefs(2,4) Gefs(3,4)];
M=[M1;M2;M3;M4;M5;M6;Mef];
end
```

Funcţia traiectorie

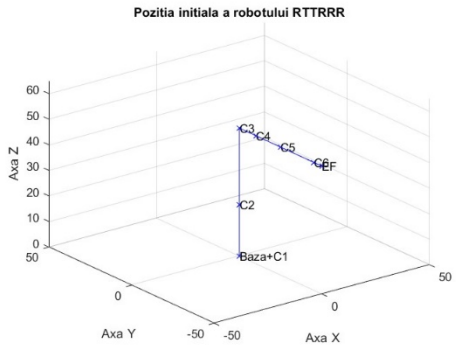
```
function []=traiectorie(l1,G,pas,fs,p,C,Cef,etichete)
% calculul pozitiei robotului pentru o miscare in ficare cupla +
% traiectorie + sistemele de referinta
figure;
q1f=input('valoare finala in cupla 1 q1f= ');
q2f=input('valoare finala in cupla 2 q2f= ');
q3f=input('valoare finala in cupla 3 q3f= ');
q4f=input('valoare finala in cupla 4 q4f= ');
q5f=input('valoare finala in cupla 5 q5f= ');
q6f=input('valoare finala in cupla 6 q6f= ');
q1v=linspace(0,q1f,pas);
q2v=linspace(0,q2f,pas);
q3v=linspace(0,q3f,pas);
q4v=linspace(0,q4f,pas);
q5v=linspace(0,q5f,pas);
q6v=linspace(0,q6f,pas);
for i=1:pas
   q1=q1v(i);
   q2=q2v(i);
   q3=q3v(i);
   q4=q4v(i);
   q5=q5v(i);
   q6=q6v(i);
   Gefs=subs(G.Gef);
   Mef(i,1)=Gefs(1,4);
   Mef(i,2)=Gefs(2,4);
   Mef(i,3)=Gefs(3,4);
end
M1=[0 0 0];
M2=[0 0 l1+q2];
```

```
G20s=subs(G.G20);
M3=[G20s(1,4) G20s(2,4) G20s(3,4)];
G30s=subs(G.G30);
M4=[G30s(1,4) G30s(2,4) G30s(3,4)];
G50s=subs(G.G50);
M5=[G50s(1,4) G50s(2,4) G50s(3,4)];
G60s=subs(G.G60);
M6=[G60s(1,4) G60s(2,4) G60s(3,4)];
Gefs=subs(G.Gef);
MMef=[Gefs(1,4) Gefs(2,4) Gefs(3,4)];
MM=[M1;M2;M3;M4;M5;M6;MMef];
% orientarea axelor pt pozitia finala
nf=[Gefs(1,1) Gefs(2,1) Gefs(3,1)];
of=[Gefs(1,2) Gefs(2,2) Gefs(3,2)];
af=[Gefs(1,3) Gefs(2,3) Gefs(3,3)];
p.xf=MMef+nf*fs;
p.yf=MMef+of*fs;
p.zf=MMef+af*fs;
% grafice
% grafic pozitie initiala
plot3(C(:,1),C(:,2),C(:,3),'bx-');
hold on;
% grafic pozitie finala
plot3(MM(:,1),MM(:,2),MM(:,3),'rx-');
% grafic traiectorie
plot3(Mef(:,1),Mef(:,2),Mef(:,3),'yo-');
% grafice axele CS initial
plot3([Cef(1) p.xi(1)],[Cef(2) p.xi(2)],[Cef(3) p.xi(3)],'b','LineWidth',3);
plot3([Cef(1) p.yi(1)],[Cef(2) p.yi(2)],[Cef(3) p.yi(3)],'r','LineWidth',3);
plot3([Cef(1) p.zi(1)],[Cef(2) p.zi(2)],[Cef(3) p.zi(3)],'g','LineWidth',3);
% grafice axele CS final
plot3([MMef(1) p.xf(1)],[MMef(2) p.xf(2)],[MMef(3)
p.xf(3)],'b','LineWidth',3);
plot3([MMef(1) p.yf(1)],[MMef(2) p.yf(2)],[MMef(3)
p.yf(3)],'r','LineWidth',3);
plot3([MMef(1) p.zf(1)],[MMef(2) p.zf(2)],[MMef(3)
p.zf(3)],'g','LineWidth',3);
grid on;
title('Pozitia si traiectoria robotului RTTRRR cu miscare in fiecare cupla');
xlabel('Axa X');
```

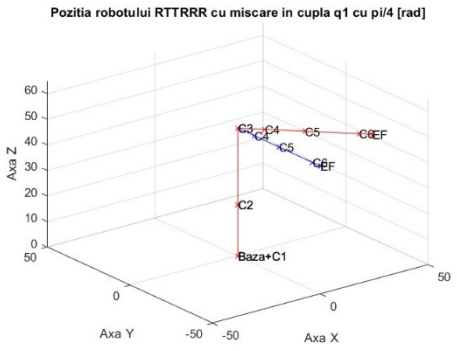
```
ylabel('Axa Y');
zlabel('Axa Z');
xlim([-50 50]);
ylim([-50 50]);
zlim([0 65]);
text(C(:,1),C(:,2),C(:,3),etichete);
text(MM(:,1),MM(:,2),MM(:,3),etichete);
end
```

Lungimile elementelor sunt declarate la începutul programului principal. Pentru verificare se va reprezenta grafic poziția inițială, apoi se va suprapune peste poziția inițială și poziția finală pentru o mișcare de pi/4 radiani în fiecare cuplă de rotație și câte 10 centimetri pentru cuplele de translație. Pentru validare, miscarea se va realiza pentru fiecare cuplă în parte (Fig. 3.4). Apoi pentru stabilirea traiectoriei se vor introduce de la tastatura valoarea finală pentru fiecare cuplă. In continuare sunt prezentate valorile introduse de la tastatură:

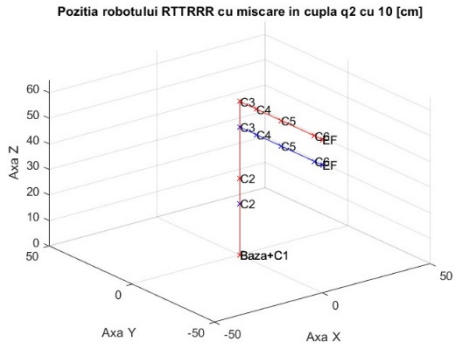
valoare finala in cupla 1 q1f= pi/2
valoare finala in cupla 2 q2f= 5
valoare finala in cupla 3 q3f= 10
valoare finala in cupla 4 q4f= pi/4
valoare finala in cupla 5 q5f= -pi/2
valoare finala in cupla 6 q6f= pi



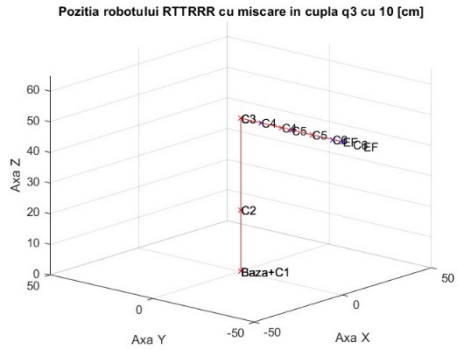
a) pozitia initială



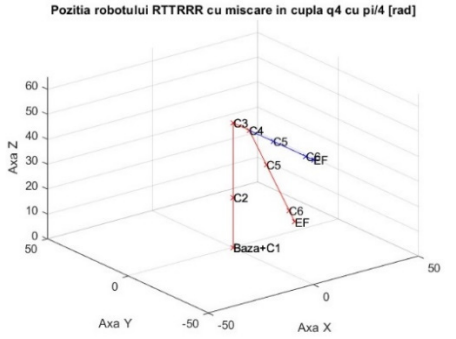
b) mișcarea din prima cuplă



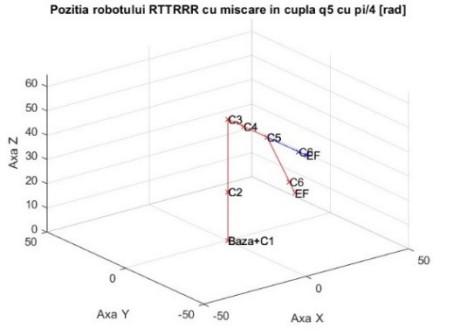
c) mişcarea în a doua cuplă



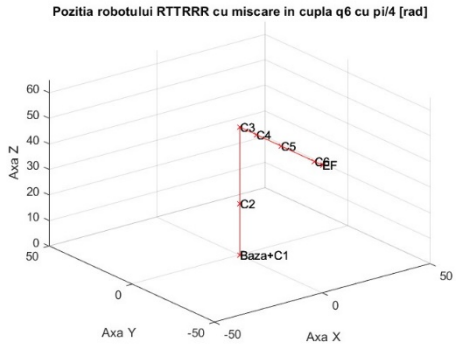
d) mişcarea în a treia cuplă



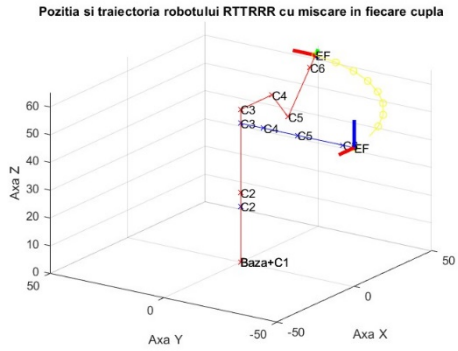
e) mişcarea în a patra cuplă



f) mişcarea în a cincea cuplă



g) mişcarea în a şasea cuplă



h) traiectoria

Fig. 3.4. Grafice model geometric direct

3.4 Aplicație 4 – Procesarea datelor experimentale

S-au realizat mai multe încercări de compresiune pe diferite epruvete din material ceramic. Echipamentul utilizat pentru testarea epruvetelor permite exportul rezultatelor pentru fiecare epruvetă sub formă de fisier csv. Programul

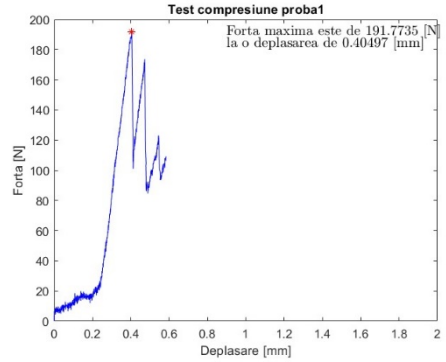
realizat va permite citirea datelor din mai multe fișiere (câte epruvete au fost testate), va realiza o reprezentarea grafică a forței funcție de deformație pentru fiecare epruvetă. În reprezentările grafice vor fi menționate vârfurile (peaks) și valoarea forţei maxime.

```
% program determinare maxime si pickuri
% pentru teste compresiune pe materiale ceramice
% curatare
clear;
clc;
% nr de probe(cate masuratori s-au efectuat)
probe=input('introducti nr de probe = ');
% valoarea treshhold-ului pentru determinarea varfurilor
trs=input('threshold = ');
% generarea de figuri pentru grafice
for i=1:2*probe+2
nume_fig{i}=figure();
ax_name{i}=axes('Parent',nume_fig{i});
end
% citire date intrare
for i=1:probe
   filename=strcat('Specimen_RawData_',num2str(i),'.csv');
   ceramica_initial{i}=csvread(filename,12,0);
   labels{i}=strcat('proba_',num2str(i));
end
%aranjare valori
for k=1:probe
   ceramica=ceramica_initial{k};
   j=1;
   last=size(ceramica);
   while ceramica(j,3)>=(-10)
      j=j+1;
   end
   cv=ceramica(j:last(1),2:3)*(-1);
   last=size(cv);
   x=cv(1,1);
   for i=1:last(1);
      cv(i,1)=cv(i,1)-x;
   end
   cv(1,2)=0;
   ceramica_final{k}=cv;
```

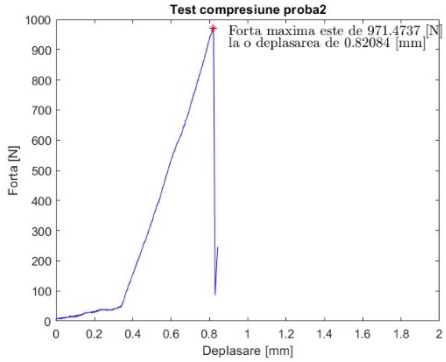
```
end
% afisare fiecare grafic individual
for i=1:probe
   axes(ax_name{i});
   tablou=ceramica_final{i};
   dep=tablou(:,1);
   forta=tablou(:,2);
   [M poz]=max(forta);
   textstr1=['Forta maxima este de ',num2str(M),' [N]'];
   textstr2=['la o deplasarea de ',num2str(dep(poz)),' [mm]'];
   titlu=strcat('Test compresiune proba ',num2str(i));
   plot(dep,forta,'b-',dep(poz),M,'r*');
   title(titlu);
   xlabel('Deplasare [mm]');
   ylabel('Forta [N]');
   xlim([0 2]);
   text('string',textstr1,'interpreter','latex','fontsize',12,'units','norm','pos',[.45
.96]);
   text('string',textstr2,'interpreter','latex','fontsize',12,'units','norm','pos',[.45
.92]);
   print(nume_fig{i},'-dpsc','-append','rezultate.ps');
   maxime(i,1)=dep(poz);
   maxime(i,2)=M;
end
% afisare fiecare grafic individual cu peaks
for i=1:probe
   axes(ax_name{i+probe});
   tablou=ceramica_final{i};
   dep=tablou(:,1);
   forta=tablou(:,2);
   [M poz]=max(forta);
   [peaks locc]=findpeaks(forta,'threshold',trs);
   if length(peaks)>=1
   for ww=1:length(locc)
      abc(ww)=dep(locc(ww));
   end
   textstr1=['Forta maxima este de ',num2str(M),' [N]'];
   textstr2=['la o deplasarea de ',num2str(dep(poz)),' [mm]'];
   titlu=strcat('Test compresiune proba ',num2str(i));
   plot(dep,forta,'b-',dep(poz),M,'r*',abc,peaks,'gx');
```

```
    title(titlu);
    xlabel('Deplasare [mm]');
    ylabel('Forta [N]');
    xlim([0 2]);
    text('string',textstr1,'interpreter','latex','fontsize',12,'units','norm','pos',[.45
.96]);
    text('string',textstr2,'interpreter','latex','fontsize',12,'units','norm','pos',[.45
.92]);
    print(nume_fig{i+probe},'-dpsc','-append','rezultate.ps');
    nume_proba=strcat('P',num2str(i));
    dlmwrite('peaks.xls',nume_proba,'delimiter',',','-append');
    dlmwrite('peaks.xls',peaks,'delimiter',',','-append');
    dlmwrite('peaks.xls',abc,'delimiter',',','-append');
    clear abc;
    end
    end
% afisare deplsari la forta maxima
axes(ax_name{2*probe+1});
bar(maxime(:,1),0.25,'FaceColor','r');
set(gca, 'XTick',1:probe, 'XTickLabel',labels);
title('Valorile deplsarilor pentru valoarea maxima a fortei de compresiune');
textstr=strcat('Mean = ',num2str(mean(maxime(:,1))),' [mm]');
text('string',textstr,'interpreter','latex','fontsize',12,'units','norm','pos',[.6 .96]);
print(nume_fig{2*probe+1},'-dpsc','-append','rezultate.ps');
%afisare valori forta maxima
axes(ax_name{2*probe+2});
bar(maxime(:,2),0.5,'FaceColor','g');
set(gca, 'XTick',1:probe, 'XTickLabel',labels);
title('Valorile maxime ale fortei de compresiune');
textstr=strcat('Mean = ',num2str(mean(maxime(:,2))),' [N]');
text('string',textstr,'interpreter','latex','fontsize',12,'units','norm','pos',[.6 .96]);
print(nume_fig{2*probe+2},'-dpsc','-append','rezultate.ps');
clear;
clc;
close all;
```

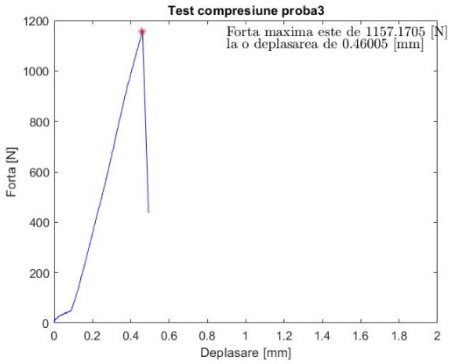
În Fig. 3.5 sunt prezentate graficele forță funcție de deplasare pentru fiecare epruvetă precum și forța de rupere. În Fig. 3.6 sunt de asemenea prezentate graficele forță funcție de deformație doar că sunt evidențiate toate vârfurile (peaks), acest lucru evidențiind propagarea fisurilor.



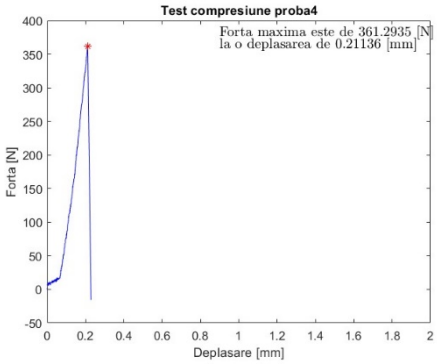
a) Epruveta 1



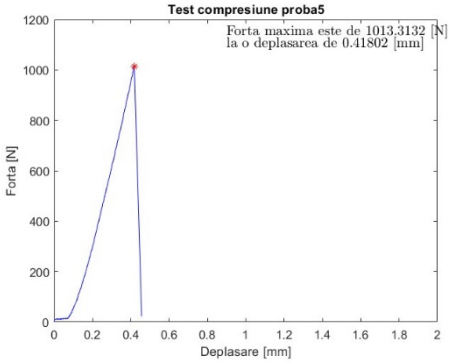
b) Epruveta 2



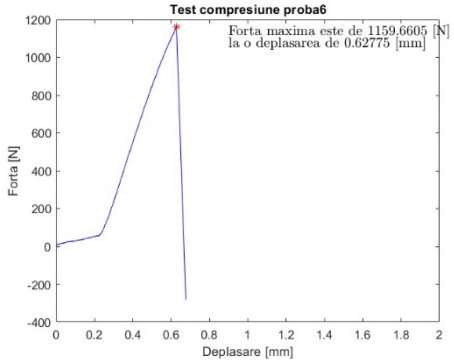
c) Epruveta 3



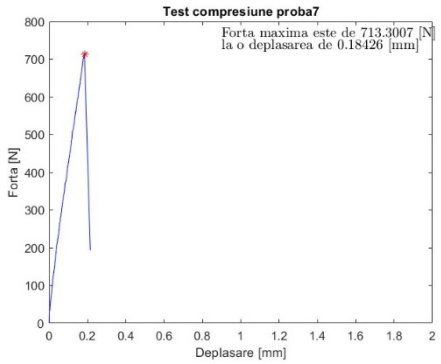
d) Epruveta 4



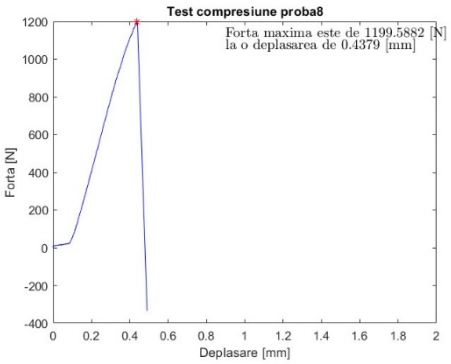
e) Epruveta 5



f) Epruveta 6

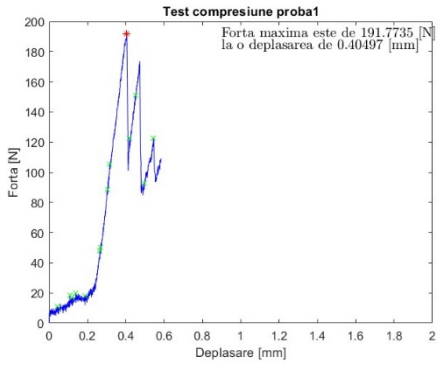


g) Epruveta 7

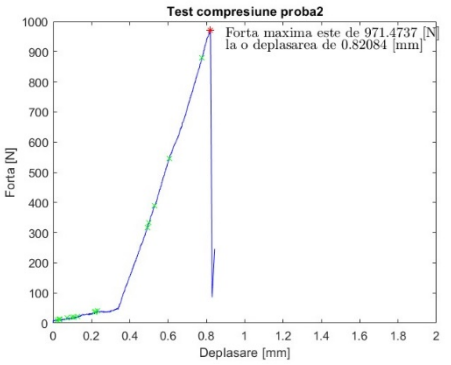


h) Epruveta 8

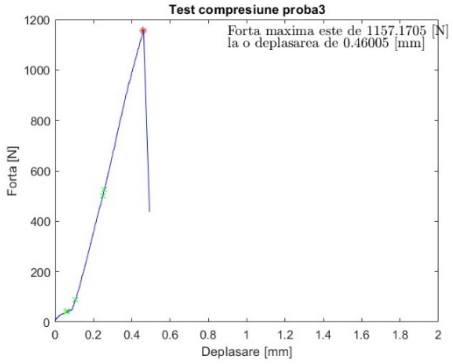
Fig.3.5. Grafice Fortă – Deformație cu reprezentarea forței de rupere



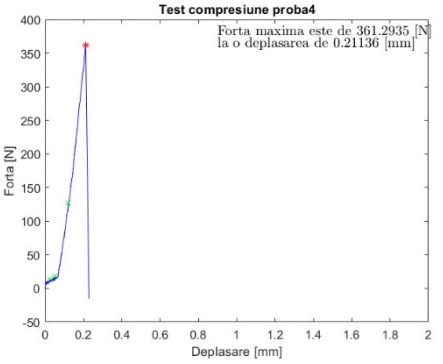
a) Epruveta 1



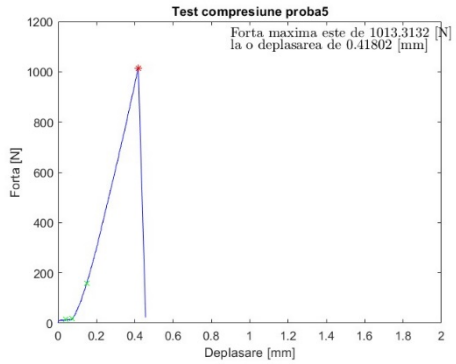
b) Epruveta 2



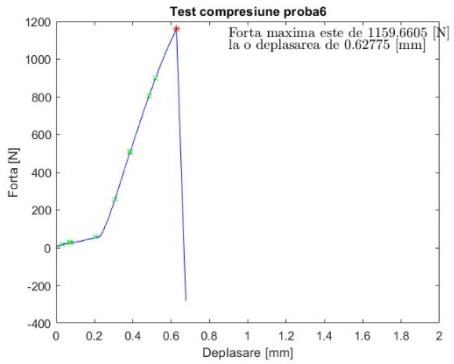
c) Epruveta 3



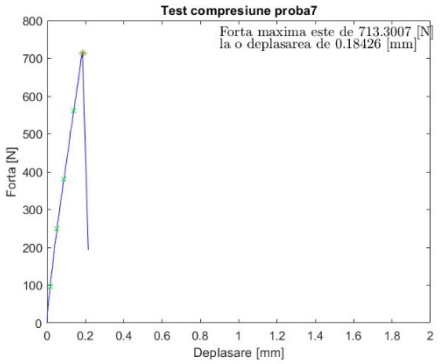
d) Epruveta 4



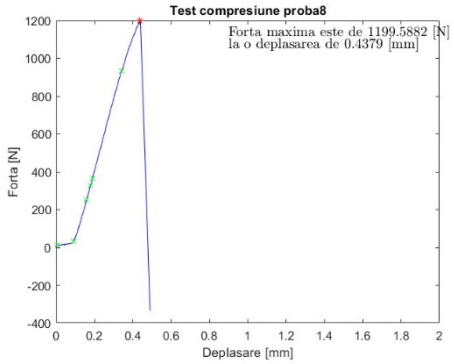
e) Epruveta 5



f) Epruveta 6



g) Epruveta 7



h) Epruveta 8

Fig.3.6 Garafice Forță – Deformație cu reprezentarea tuturor vârfurilor (peaks)

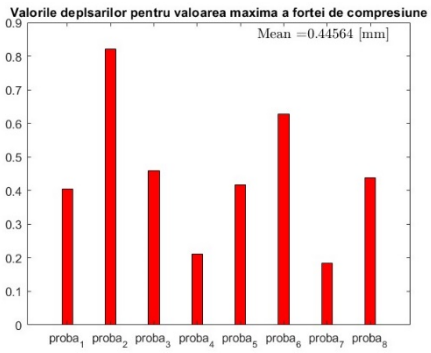


Fig. 3.7 Deformația la rupere

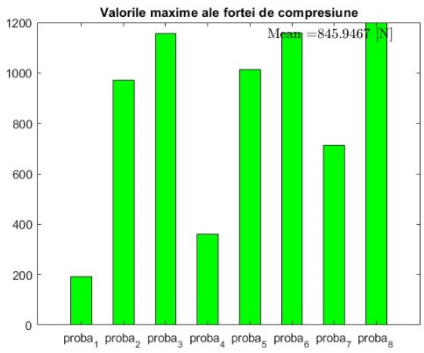


Fig. 3.8 Forta de rupere

În Fig. 3.7 și Fig. 3.8 sunt centralizate rezultatele deformației și forței la rupere.

Calculele realizate, adică, găsirea vârfurilor, forțelor maxime, deplasari la forța maximă, și valorile medi ale acestora vor fi salvate într-un fișier excel, iar reprezenările grafice vor fi de asemenea salvate în fișier postscript.

3.5 Aplicație 5 - Interfață grafică calculator

Se va realiza o aplicație (Fig. 3.9) utilizând modulul appdesigner care să simuleze un calculator de buzunar.

În figura este prezentată intefața grafică utilizator creată. Ea conține:

- un titlu (obiect tip label);
- un ecran (obiect de tip EditFieldText);
- butoane pentru fiecare cifră și operație;
- buton de închidere;
- meniu pentru selectarea culorii textului din ecran.

Fiecarui buton i se va atașa o funcție care va rula la apăsarea acestuia. Pentru fieacare cifră, operație respectiv paranteză funcția trebuie să adauge caracterul corespunzător la șirul de caractare existent pe ecran. Astfel funcțiile pentru aceste butoane sunt similare, și se va exemplifica doar pentru caracterul '1'. Celelalte funcții sunt exemplificate individual.

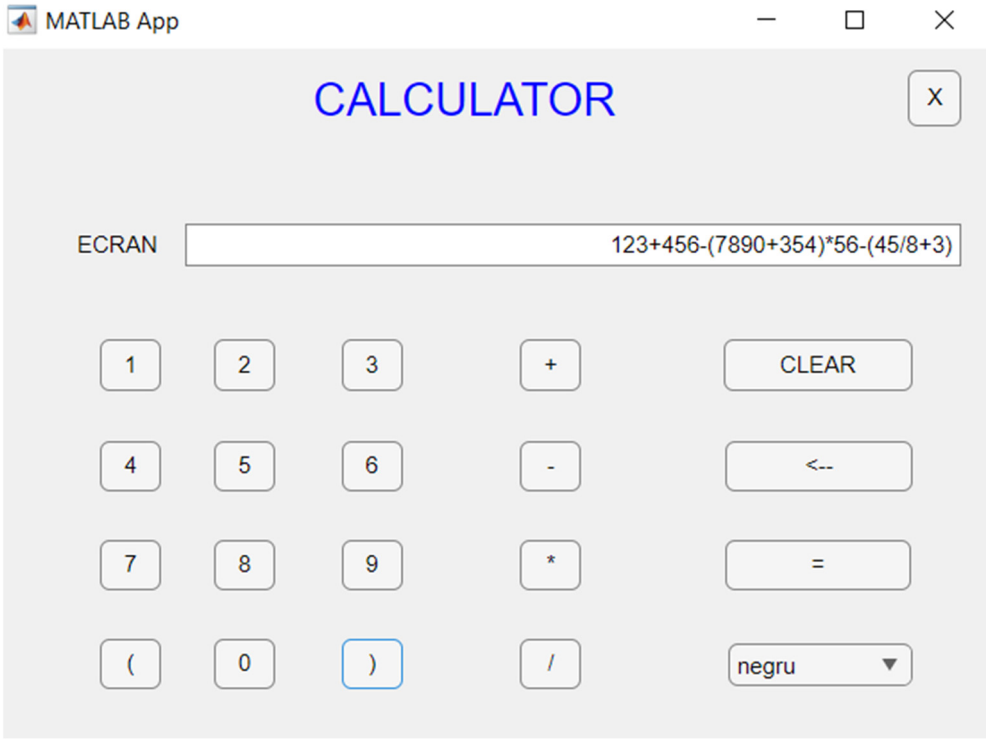


Fig. 3.9. Interfață grafică calculator de buzunar

```
% Button pushed function: Add 1
function ButtonPushed(app, event)
        % prin comnda strcat se adauga caracterul 1 la capatul sirului existent
        app.ECRANEditField.Value=strcat(app.ECRANEditField.Value,'1');
end
% Button pushed function: CLEAR
function CLEARButtonPushed(app, event)
        % valoarea noua a sirului va fi blank
        app.ECRANEditField.Value='';
end
% Button pushed function: ← backspace
function Button_16Pushed(app, event)
        % pentru stergere se va reinitializa sirul cu valoarea sa minus ultimul
        % caracter
        app.ECRANEditField.Value=app.ECRANEditField.Value(1:end-1);
end
% Button pushed function: = egal
function Button_17Pushed(app, event)
        % butonul = va evalua textul scris pe ecran rezultatul fiind un numar
        % apoi va converti numarul in text pentru a putea fi afisat
    app.ECRANEditField.Value=num2str(eval(app.ECRANEditField.Value));
end
% Button pushed function: X inchidere
function XButtonPushed(app, event)
        % comanda closereq este utilizata pentru inchiderea ferestrei curente
        closereq();
end
% Value changed function: CuloareDropDown
function CuloareDropDownValueChanged(app, event)
        % se selectează o culoarea
        switch str2num(app.CuloareDropDown.Value)
                case 1
                        culoare='black';
                case 2
                        culoare='red';
                case 3
                        culoare='blue';
                case 4
                        culoare='green';
                case 5
```

```
                culoare='yellow';
        end
        % se formatează textul ecranului cu culoarea selectata
        app.ECRANEditField.FontColor=culoare;
end
```

3.6 Aplicație 6 – Definirea și utilizarea funcțiilor

Pe baza parametrilor introduși de la tastatură să se calculeze aria triunghiului, dreptunghiului, trapezului și cercului. Calculul ariei se va efectua printr-o funcție independentă pentru fiecare formă geometrică. Problema se va rezolva în două moduri: clasic (script) și prin realizarea unei interfețe grafice.

Modul 1 – script

```
%Programul principal
% calculul ariei triunghi, dreptunghi, trapez, cerc
% curatare cw si ws
clear;
clc;
% alegerea formei geometrice
disp('1 - Triunghi');
disp('2 - Dreptunghi');
disp('3 - Trapez');
disp('4 - Cerc');
optiune=input('Optiunea dvs este: ');
% daca otiunea este diferita de 1, 2,3 sau 4 se va cere reintroducerea acesteia
while (optiune~=1)&&(optiune~=2)&&(optiune~=3)&&(optiune~=4)
   disp('Optiune incorecta!');
   optiune=input('Optiunea dvs este: ');
end
% functie de optiunea aleasa se apeleaza functia corespunzatoare
switch optiune
   case 1
      arie=triunghi();
   case 2
      arie=dreptunghi();
   case 3
      arie=trapez();
   case 4
      arie=cerc();
```

```
end
% se afiseaza aria
disp(horzcat('Aria = ',num2str(arie)));
% functia de calcul al ariei triunghiului
function [a]=triunghi()
b=input('baza = ');
h=input('inaltimea = ');
a=b*h/2;
end
% functia de calcul al ariei dreptunchiului
function [a]=dreptunghi()
L=input('lungimea = ');
l=input('latimea = ');
a=L*l;
end
% functia de calcul al ariei trapezului
function [a]=trapez()
b=input('baza mica = ');
B=input('baza mare = ');
h=input('inaltimea = ');
a=(b+B)*h/2;
end
% functia de calcul al ariei cercului
function [a]=cerc()
r=input('raza = ');
a=pi*r^2;
end
```

În urma rulării acestui script în fereastra de comenzi vor aparea următorele linii:

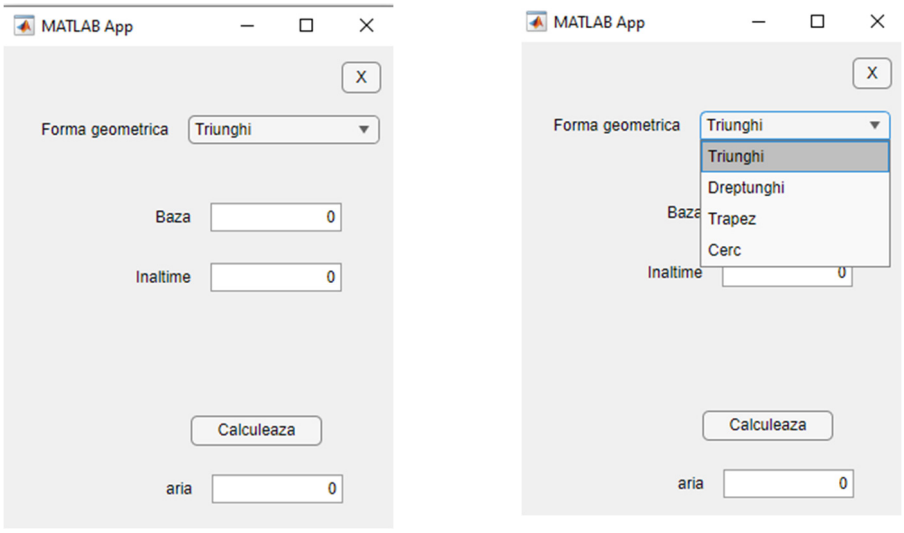
```
1 - Triunghi
2 - Dreptunghi
3 - Trapez
4 - Cerc
Optiunea dvs este: 1
baza = 23
inaltimea = 15
Aria = 172.5
```

Modul 2 – Interfață grafică utilizator

Interfața grafică (fig. 3.10) conține următoarele obiecte:

- buton închidere interfată;

- meniu de selecție a formei geometrice;
- 1, 2 sau 3 casuțe pentru introducerea parametrilor
- buton de calcul al ariei;
- câmp pentru afișarea ariei calculate.



a) design interfață b) interfața grafică (rulat)

Fig. 3.10. Interfață grafică – Calcul arii

Codul sursă al interfeței grafice este prezentat în continuare

```
% functia de calcul al ariei triunghiului
function arie = triunghi(app,b,h)
        arie=b*h/2;
end
% functia de calcul al ariei dreptunghiului
function arie = dreptunghi(app,L,l)
        arie=L*l;
end
% functia de calcul al ariei trapezului
function arie = trapez(app,b,B,h)
        arie=(b+B)*h/2;
end
% functia de calcul al ariei cercului
function arie = cerc(app,r)
        arie=pi*r^2;
end
```

```
% functia de pornire
function startupFcn(app)
        % obiectele dorite se seteaza ca vizibile si se eticheteaza
        app.var1EditFieldLabel.Text='Baza';
        app.var1EditFieldLabel.Visible="on";
        app.var2EditFieldLabel.Text='Inaltime';
        app.var2EditFieldLabel.Visible="on";
        app.var3EditFieldLabel.Visible="off";
        app.var1EditField.Visible="on";
        app.var2EditField.Visible="on";
        app.var3EditField.Visible="off";
end
% meniul de alegere al formei geometrice
function FormageometricaDropDownValueChanged(app, event)
        value = str2num(app.FormageometricaDropDown.Value);
        % functie de optiunea aleasă obiectele dorite se formateaza
        switch value
                case 1
                        app.var1EditFieldLabel.Text='Baza';
                        app.var1EditFieldLabel.Visible="on";
                        app.var2EditFieldLabel.Text='Inaltime';
                        app.var2EditFieldLabel.Visible="on";
                        app.var3EditFieldLabel.Visible="off";
                        app.var1EditField.Visible="on";
                        app.var2EditField.Visible="on";
                        app.var3EditField.Visible="off";
                case 2
                        app.var1EditFieldLabel.Text='Lungime';
                        app.var1EditFieldLabel.Visible="on";
                        app.var2EditFieldLabel.Text='Latime';
                        app.var2EditFieldLabel.Visible="on";
                        app.var3EditFieldLabel.Visible="off";
                        app.var1EditField.Visible="on"
                        app.var2EditField.Visible="on";
                        app.var3EditField.Visible="off";
                case 3
                        app.var1EditFieldLabel.Text='Baza mica';
                        app.var1EditFieldLabel.Visible="on";
                        app.var2EditFieldLabel.Text='Baza mare';
                        app.var2EditFieldLabel.Visible="on";
```

```
                app.var3EditFieldLabel.Text='Inaltimea';
                app.var3EditFieldLabel.Visible="on";
                app.var1EditField.Visible="on";
                app.var2EditField.Visible="on";
                app.var3EditField.Visible="on";
            case 4
                app.var1EditFieldLabel.Text='Raza';
                app.var1EditFieldLabel.Visible="on";
                app.var2EditFieldLabel.Visible="off";
                app.var3EditFieldLabel.Visible="off";
                app.var1EditField.Visible="on";
                app.var2EditField.Visible="off";
                app.var3EditField.Visible="off";
            end
        % se initializeaza cu zero campurile pentru introducerea datelor
        app.var1EditField.Value=0;
        app.var2EditField.Value=0;
        app.var3EditField.Value=0;
end
% butonul calculeaza
function CalculeazaButtonPushed(app, event)
        value = str2num(app.FormageometricaDropDown.Value);
        % functie de optiunea aleasa se apeleaza functia corespunzatoare
        switch value
        case 1
            arie=app.triunghi(app.var1EditField.Value,…
            app.var2EditField.Value);
            app.ariaEditField.Value=arie;
        case 2
            arie=app.dreptunghi(app.var1EditField.Value,…
            app.var2EditField.Value);
            app.ariaEditField.Value=arie;
        case 3
            arie=app.trapez(app.var1EditField.Value,…
            app.var2EditField.Value,…
            app.var3EditField.Value);
            app.ariaEditField.Value=arie;
        case 4
            arie=app.cerc(app.var1EditField.Value);
            app.ariaEditField.Value=arie;
```

```
        end
end
% buton de inchidere
function XButtonPushed(app, event)
        closereq();
end
```

3.7 Aplicație 7 – Interfață formatare grafic funcție polinomială

Se va realiza o interfață grafică utilizator pentru reprezenta grafică a unei funcții polinomiale de ordinul trei.

Interfața grafică conține următoarele obiecte:

- titlu (Label);
- buton închidere (Push Button);
- zona grafică (Axes);
- zona de introducere a parametrilor (Tab Group).
    - coeficienți polinomului (fig. 3.11);
        - coeficienți polinomului de ordinul 3;
        - intervalul de calcul al funției ([Xmin;Xmax]);
        - numărul de puncte în care se calculează funcția;
        - buton afișare grafic.
    - formatare grafic (fig. 3.12);
        - tip linie;
        - culoare linie;
        - marcaj punct;
        - grosime linie;
        - grid;
        - scara abscisă.
    - Etichetare (fig. 3.13).
        - titlu;
        - etichetă abscisă;
        - etichetă ordonată:
        - culoare fundal.

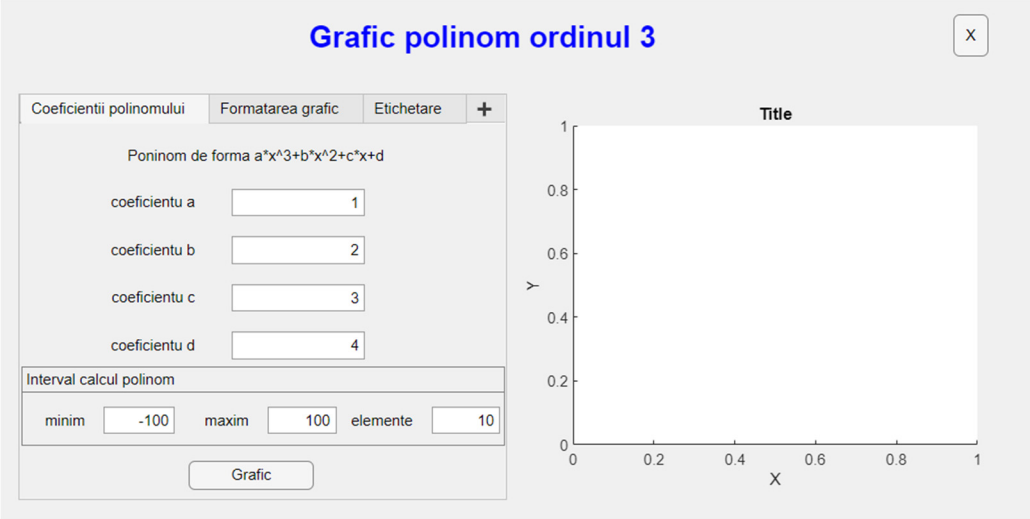


Fig. 3.11. Interfață grafică utilizator – tab Coeficienții polinomului

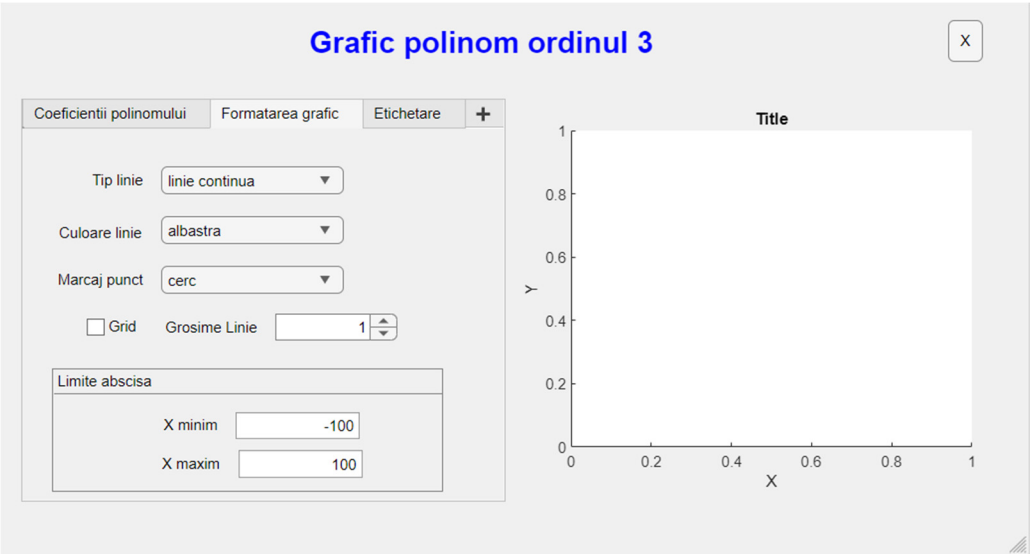


Fig. 3.12. Interfață grafică utilizator – tab Formatare grafic

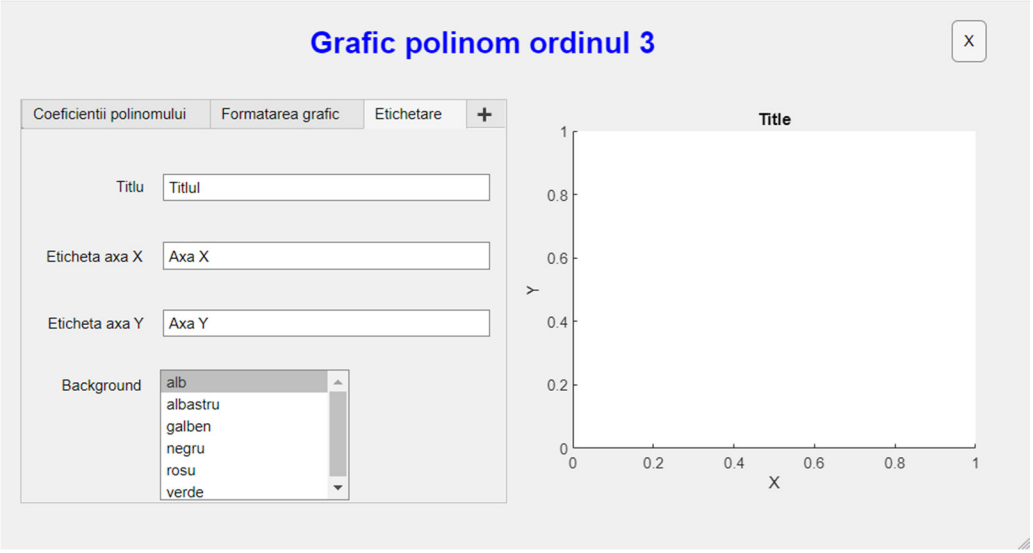


Fig. 3.13. Interfață grafică utilizator – tab Etichetare

```
% Button pushed function: GraficButton
function GraficButtonPushed(app, event)
coeficienti=[app.coeficientu_a.Value,app.coeficientu_b.Value,…
app.coeficientu_c.Value,app.coeficientu_d.Value];
x=linspace(app.minimEditField.Value,app.maximEditField.Value,…
app.elementeEditField.Value);
y=polyval(coeficienti,x);
plot(app.UIAxes,x,y);
app.UIAxes.Title.String=app.TitluEditField.Value;
app.UIAxes.XLabel.String=app.EtichetaaxaXEditField.Value;
app.UIAxes.YLabel.String=app.EtichetaaxaYEditField.Value;
app.UIAxes.XLim=[app.XminimEditField.Value,…
app.XmaximEditField.Value];
app.UIAxes.Children(1).Color=app.CuloarelinieDropDown.Value;
app.UIAxes.Children(1).LineStyle=app.TiplinieDropDown.Value;
app.UIAxes.Children(1).Marker=app.MarcajpunctDropDown.Value;
app.UIAxes.Children(1).LineWidth=app.GrosimeLinieSpinner.Value;
app.UIAxes.Color=app.BackgroundListBox.Value;
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        closereq();
end
```

```
% Value changed function: TiplinieDropDown
function TiplinieDropDownValueChanged(app, event)
        app.UIAxes.Children(1).LineStyle=app.TiplinieDropDown.Value;
end
% Value changed function: CuloarelinieDropDown
function CuloarelinieDropDownValueChanged(app, event)
        app.UIAxes.Children(1).Color=app.CuloarelinieDropDown.Value;
end
% Value changed function: MarcajpunctDropDown
function MarcajpunctDropDownValueChanged(app, event)
        app.UIAxes.Children(1).Marker=app.MarcajpunctDropDown.Value;
end
% Value changed function: XminimEditField
function XminimEditFieldValueChanged(app, event)
        app.UIAxes.XLim=[app.XminimEditField.Value,
        app.XmaximEditField.Value];
end
% Value changed function: XmaximEditField
function XmaximEditFieldValueChanged(app, event)
        app.UIAxes.XLim=[app.XminimEditField.Value,
        app.XmaximEditField.Value];
end
% Value changed function: TitluEditField
function TitluEditFieldValueChanged(app, event)
        app.UIAxes.Title.String=app.TitluEditField.Value;
end
% Value changed function: EtichetaaxaXEditField
function EtichetaaxaXEditFieldValueChanged(app, event)
        app.UIAxes.XLabel.String=app.EtichetaaxaXEditField.Value;
end
% Value changed function: EtichetaaxaYEditField
function EtichetaaxaYEditFieldValueChanged(app, event)
        app.UIAxes.YLabel.String=app.EtichetaaxaYEditField.Value;
end
% Value changed function: GridCheckBox
function GridCheckBoxValueChanged(app, event)
        if app.GridCheckBox.Value==1
                grid (app.UIAxes,'on');
        else
                grid (app.UIAxes,'off');
```

```
        end
end
% Value changed function: GrosimeLinieSpinner
function GrosimeLinieSpinnerValueChanged(app, event)
        app.UIAxes.Children(1).LineWidth=app.GrosimeLinieSpinner.Value;
end
% Value changed function: BackgroundListBox
function BackgroundListBoxValueChanged(app, event)
        app.UIAxes.Color=app.BackgroundListBox.Value;
end
```

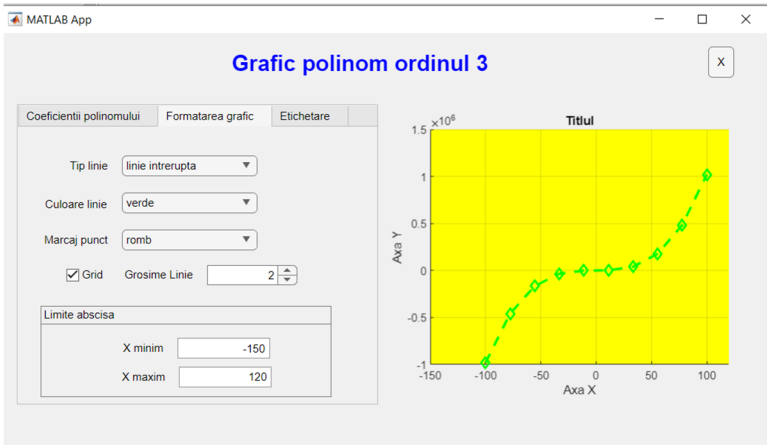


Fig. 3.14. Formatare grafic

Rezultatul final în urma formatării graficului este prezentat în Fig. 3.14.

3.8 Aplicație 8 - Gestionarea unei baze de date

Pentru a exemplifica modul de gestionare al unei baze de date utilizând modulul databasetoolbox s-a creat o bază de date al unui cabinet veterinar. Aplicația a fost creată local astfel pentru a putea accesa baza de date a fost necesară instalarea aplicației **xampp** (Fig. 3.15), care pune la dispoziție modulele apache, mysql, php și phpMyAdmin necesare pentru crearea bazei de date [4].

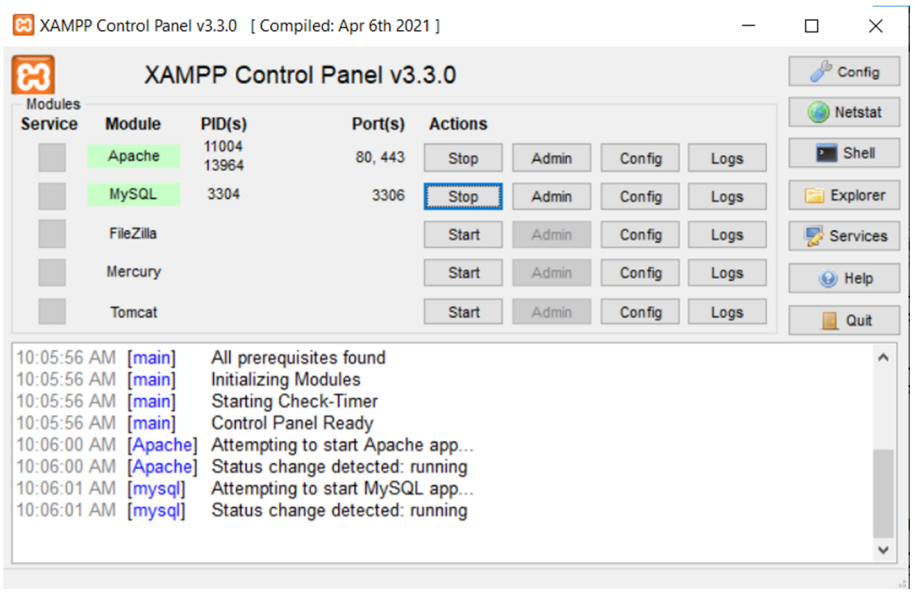


Fig. 3.15. Aplicația XAMPP

După pornirea serverelor apache și mysql este necesar configurarea aplicației phpMyAdmin deoarece inițial accesul la baza de date se realizează fără parolă. Acest lucru se realizează accesând fișierul de configurare (Fig. 3.16)

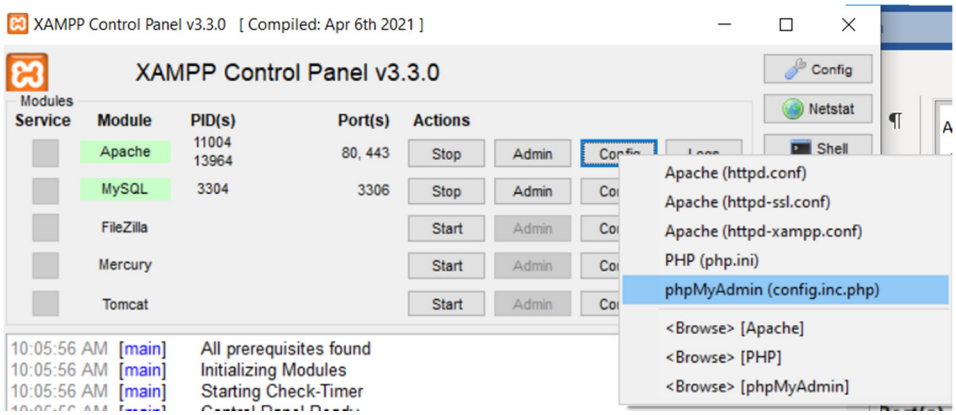


Fig. 3.16. Fișier configurare phpMyAdmin

În fișierul config.inc.php (Fig. 3.17) trebuie setat ca tipul autentificări (’auth_type’) să fie **cookie.**

config.inc.php - Notepad

File Edit Format View Help

```
$i++;

/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'cookie';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = true;
$cfg['Lang'] = '';
```

Fig. 3.17. Fișier configurare php

De asemena pentru a realiza conexiunea între Matlab și mysql este necesar instalarea unui driver. Pentru exemplul de față s-a utilizat driverul mysql-connector-java-8.0.18. Acest driver este disponibil gratuit pe internet. Pentru instalarea acestuia trebuie copiat fișierul de tip *.jar în directorul unde este instalat programul Matlab. Locația unde trebuie copiat driver-ul este prezentată în Fig. 3.18.

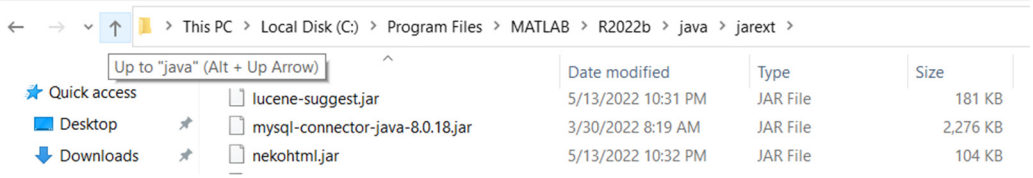


Fig.3.18. Locație driver

Pentru ca programul Matlab să poată accesa acest driver trebuie menționată numele și locația driver-ului în fișierul classpath. Locația fișierului este prezentată în Fig. 3.19. În acest fișier trebuie menționat exact numele și locația driver-ului (Fig. 3.20).

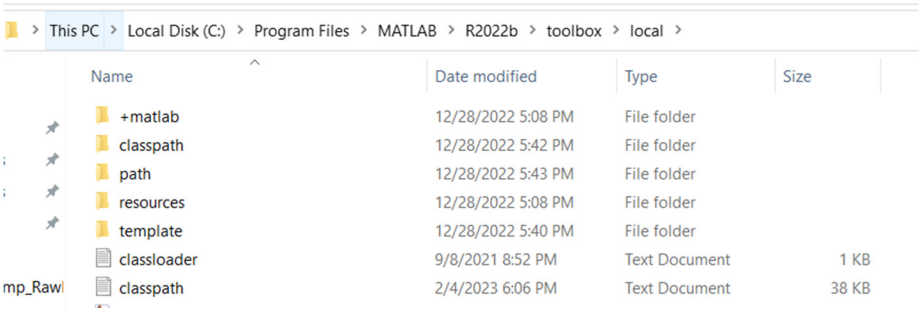


Fig. 3.19. Locație fișier class path

classpath - Notepad

File Edit Format View Help

```
$matlabroot/java/jar/zh_CN/toolbox_packaging_java_res.jar
$matlabroot/java/jar/zh_CN/trialsinstaller_res.jar
$matlabroot/java/jar/zh_CN/update_installer_res.jar
$matlabroot/java/jar/zh_CN/update_installer_shutdown_client_res.jar
$matlabroot/java/jarext/mysql-connector-java-8.0.18.jar
```

Fig. 3.20. Fişier calsspath.txt

Dacă toate etapele prezentate anterior au fost efectuate, atunci în browser la adresa localhost/phpmyadmin putem să ne logăm cu credenţialele noastre şi crea baza de date dorită (Fig. 3.21).

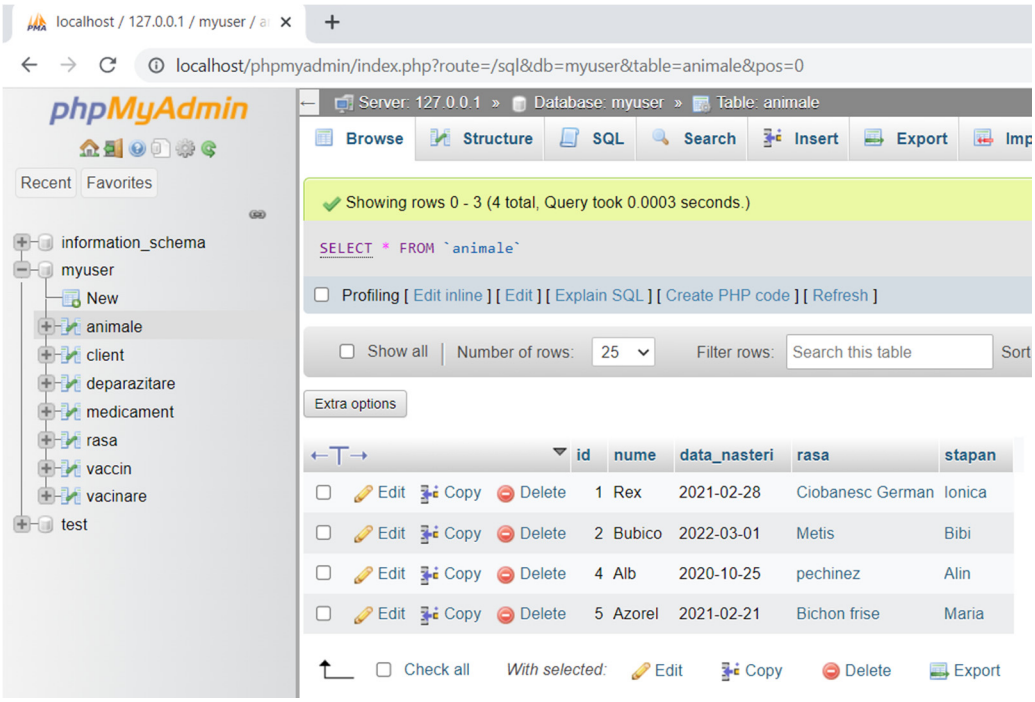


Fig. 3.21. Interfaţă phpMyAdmin

Pentru aplicaţia de faţă s-a creat o bază de date cu şapte tabele interconectate (Fig. 3.22). Tabelul principal se numeşte **animale** şi conţine lista tuturor animalelor care au fost consultate la clinică. Acesta este conectat cu tabelele **client**, **vacinare**, **deparazitare** şi **rasa**. Tabelul client conţine datele stapânului animalului, tabelul vaccinare conţine tipul vaccinului şi data

administrări acestuia, tabelul deparazitare conține medicamentele și data administrării. Tabelele **rasa**, **vaccin** și **medicament** conțin lista disponibilă pentru rasă, vaccin respectiv medicamente.

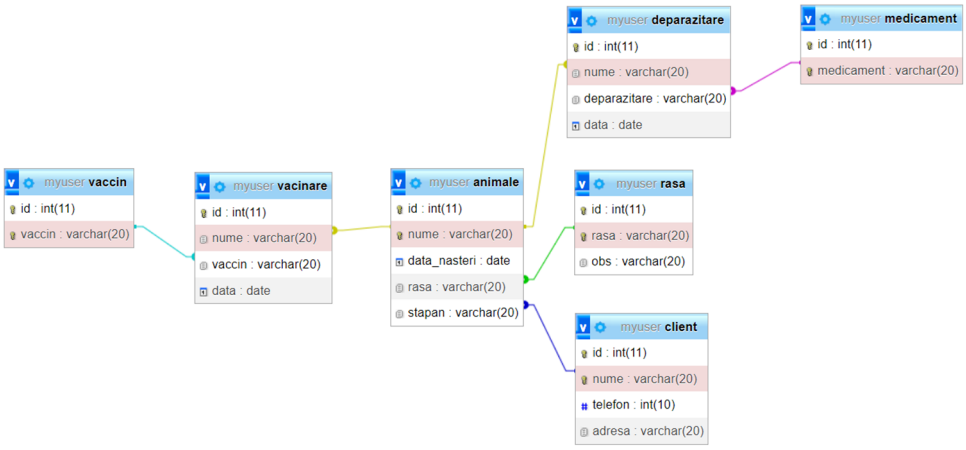


Fig. 3.22. Design-ul bazei de date

Aplicația conține șapte interfețe grafice interconectate între ele (Fig. 3.23). Pornirea aplicației se face prin rularea interfeței **conectare** care permite introducerea credențialelor pentru conectarea la baza de date. Dacă datele introduse sunt corecte atunci se deschide fereastra principală (interfața principală). Dacă datele introduse sunt incorecte atunci apare un mesaj care precizează acest lucru apoi permite reintroducerea credențialelor.

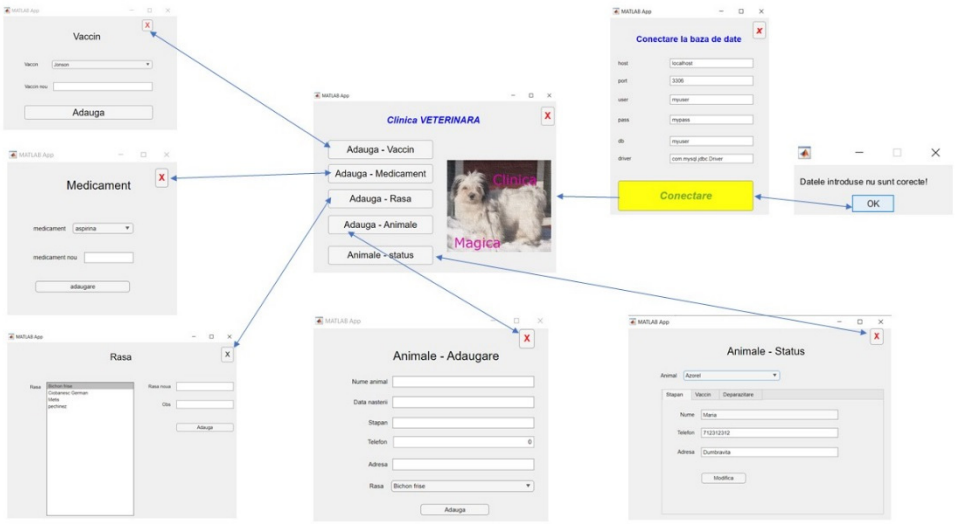


Fig. 3.23. Legăturile dintre interfețele grafice

În Fig. 3.24 se observă interfața de conectare care conține șase câmpuri editabile pentru introducerea tuturor datelor necesare logării la baza de date. După introducerea acestora prin apasarea butonului **Conectare** se realizează conexiunea la baza de date.

Fig. 3.24. Intefața de conectare la baza de date

Interfața de conectare conține două funcții. Una pentru închiderea aplicației respectiv cealaltă pentru conectarea la baza de date. În continuare sunt prezentate aceste funcții.

```
% Button pushed function: xButton
function xButtonPushed(app, event)
        closereq();
end
% Button pushed function: ConectareButton
function ConectareButtonPushed(app, event)
url=strcat('jdbc:mysql://',app.host.Value,':',app.port.Value,'/',app.db.Value);
conn=database(app.db.Value,app.user.Value,app.pass.Value,app.driver.Value,u
rl);
  assignin('base','conectare',conn);
if strcmp(conn.Driver,'com.mysql.jdbc.Driver')==1
        principala;
        closereq();
else
        msgbox('Datele introduse nu sunt corecte!');
end
        setappdata(0,'conectare',conn);
```

```
end
```

Dacă sa putut realiza conectarea la baza de date atunci se va salva variabila **conn** care conține datele conexiunii astfel încât acesta să fie disponibilă și pentru celelalte interfețe utilizator urmată de apelarea interfeței principale (Fig. 3.25).

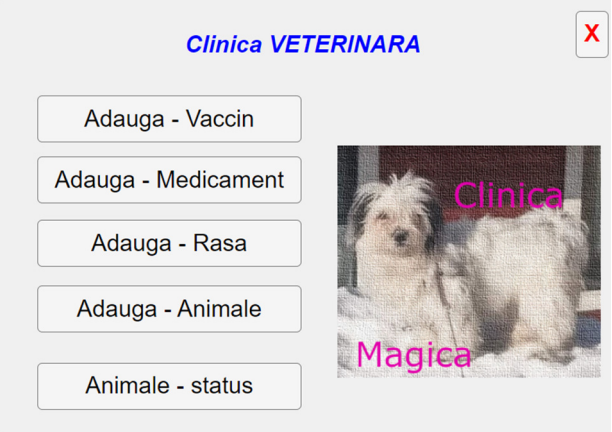


Fig. 3.25. Interfața principală

Interfața principală face defapt legatura cu toate celelalte interfețe permițând accesul la interfața dorită. Fiecare din cele cinci butoane accesează câte o interfață. Funcțiile din spatele fiecărui buton sunt prezentate în continuare.

```
properties (Access = private)
        conn % variabila de conectare la baza de date
end
% Code that executes after component creation
function startupFcn(app)
        app.conn=getappdata(0,'conectare');
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        closereq();
end
% Button pushed function: AdaugaVaccinButton
function AdaugaVaccinButtonPushed(app, event)
        vaccin;
        closereq();
end
% Button pushed function: AdaugaMedicamentButton
```

```
function AdaugaMedicamentButtonPushed(app, event)
        medicament;
        closereq();
end
% Button pushed function: AdaugaAnimaleButton
function AdaugaAnimaleButtonPushed(app, event)
        adaugare;
        closereq();
end

% Button pushed function: AnimalestatusButton
function AnimalestatusButtonPushed(app, event)
        status;
        closereq();
end
% Button pushed function: AdaugaRasaButton
function AdaugaRasaButtonPushed(app, event)
        rasa;
        closereq();
end
```

În Fig. 3.26 este prezentată interfața **Vaccin** care permite adăugarea în tabelul cu vaccinuri, a unui nou vaccin care devine disponibil pentru administrare.

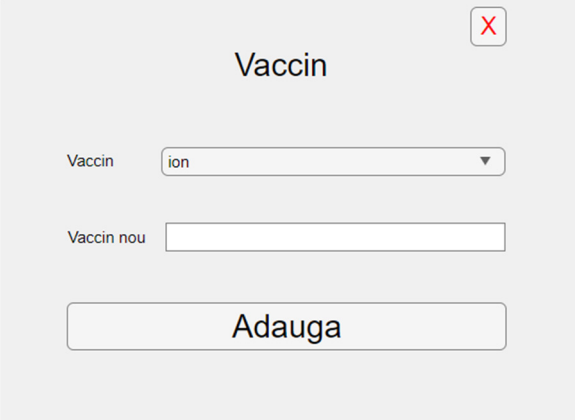


Fig. 3.26. Interfața vaccin

Pentru a realiza adaugarea unui nou vaccin în listă se va utiliza urmatorul cod sursă.

```
properties (Access = private)
        conn % variabila de conectare la baza de date
end
% Code that executes after component creation
function startupFcn(app)
        app.conn=getappdata(0,'conectare');
        comanda='select vaccin from vaccin';
        rezultat=fetch(app.conn,comanda);
        app.VaccinDropDown.Items=rezultat.vaccin;
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        principala;
        closereq();
end
% Button pushed function: AdaugaButton
function AdaugaButtonPushed(app, event)
        comanda=strcat('insert into vaccin (vaccin) values ('',…
        app.VaccinnouEditField.Value,'');');
        exec(app.conn,comanda);
        comanda='select vaccin from vaccin';
        rezultat=fetch(app.conn,comanda);
        app.VaccinDropDown.Items=rezultat.vaccin;
        app.VaccinnouEditField.Value='';
end
```

Se observă existența funcției de pornire (startup function) care preia variabila **conn** pentru accesul la baza de date, precum și populează popupul vaccin cu lista disponibilă a acestora.

În Fig. 3.27 este prezentată interfața ce permite adăugarea de medicamente în baza de date. Aceasta este similară interfeței **Vaccin.**

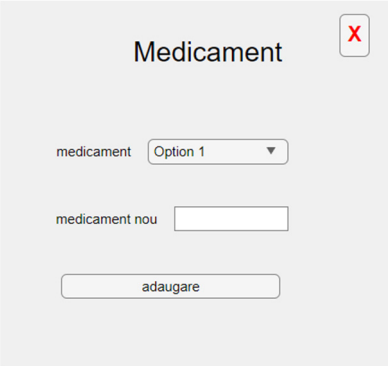


Fig. 3.27. Interfața medicament

Codul sursă pentru acestă interfață este prezentat în continuare.

```
properties (Access = private)
        conn % Description
end
% Code that executes after component creation
function startupFcn(app)
        app.conn=getappdata(0,'conectare');
        comanda='select medicament from medicament';
        rezultat=fetch(app.conn,comanda);
        app.medicamentDropDown.Items=rezultat.medicament;
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        principala;
        closereq();
end
% Button pushed function: adaugareButton
function adaugareButtonPushed(app, event)
        comanda=strcat('insert into medicament (medicament) values ('',…
        app.medicamentnouEditField.Value,'');');
        exec(app.conn,comanda);
        comanda='select medicament from medicament';
        rezultat=fetch(app.conn,comanda);
        app.medicamentDropDown.Items=rezultat.medicament;
        app.medicamentnouEditField.Value='';
end
% Value changed function: medicamentDropDown
function medicamentDropDownValueChanged(app, event)
```

```
% app.VaccinulselectatEditField.Value = app.medicamentDropDown.Value;
end
```

Interfața care permite adăugarea unei noi rase de animal în baza de date este prezentată în Fig. 3.28.

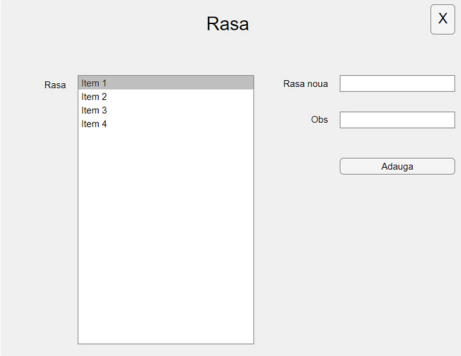


Fig. 3.28. Interfața rasă

În continuare se prezintă codul sursă pentru adaugarea unei noi rase.

```
properties (Access = private)
        conn % Description
end
% Code that executes after component creation
function startupFcn(app)
        app.conn=getappdata(0,'conectare');
        comanda='select rasa from rasa';
        rezultat=fetch(app.conn,comanda);
        app.RasaListBox.Items=rezultat.rasa;
        value = app.RasaListBox.Value;
        comanda=strcat('select obs from rasa where rasa="',value,'";');
        rezultat=fetch(app.conn,comanda);
        app.ObsEditField.Value=rezultat.obs{1};
end
% Value changed function: RasaListBox
function RasaListBoxValueChanged(app, event)
        value = app.RasaListBox.Value;
        comanda=strcat('select obs from rasa where rasa="',value,'";');
        rezultat=fetch(app.conn,comanda);
        app.ObsEditField.Value=rezultat.obs{1};
end
```

```
% Button pushed function: AdaugaButton
function AdaugaButtonPushed(app, event)
        comanda=strcat('insert into rasa (rasa,obs) values ('',…
        app.RasanouaEditField.Value,'','',app.ObsEditField.Value,'');');
        exec(app.conn,comanda);
        comanda='select rasa from rasa';
        rezultat=fetch(app.conn,comanda);
        app.RasaListBox.Items=rezultat.rasa;
        value = app.RasaListBox.Value;
        comanda=strcat('select obs from rasa where rasa='',value,'';');
        rezultat=fetch(app.conn,comanda);
        app.ObsEditField.Value=rezultat.obs{1};
        app.RasanouaEditField.Value='';
end
function XButtonPushed(app, event)
        principala;
        closereq();
end
```

În Fig. 3.29 este prezentată interfața prentru adaugarea unui nou animal de companie. Pe langă informațiile despre animal, mai trebuie introduse și informațiile despre stapânul animalului de companie.

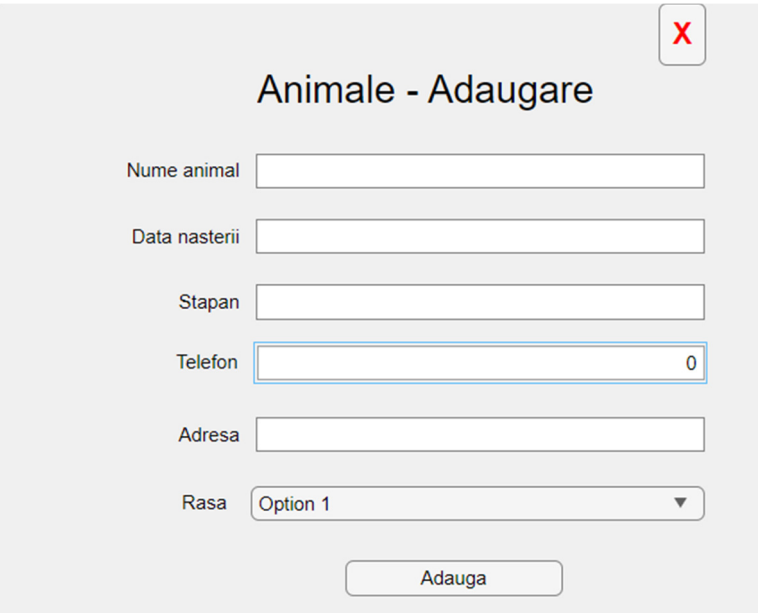


Fig. 3.29. Interfața animal de companie

Datele introduse prin acestă interfață se vor salva în tabelele corespunătoare animalelor de companie respectiv clienților. Liniile de comandă care realizează acest lucru sunt prezentate in continuare.

```
properties (Access = private)
        conn % Description
        end
% Code that executes after component creation
function startupFcn(app)
        app.conn=getappdata(0,'conectare');
        comanda='select rasa from rasa';
        rezultat=fetch(app.conn,comanda);
        app.RasaDropDown.Items=rezultat.rasa;
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        principala;
        closereq();
end
% Button pushed function: AdaugaButton
function AdaugaButtonPushed(app, event)
        comanda=strcat('insert into client (nume,telefon,adresa) values ('',…
        app.StapanEditField.Value,'','',num2str(app.TelefonEditField.Value),…
        '','',app.AdresaEditField.Value,'');')
        exec(app.conn,comanda);
        comanda=strcat('insert into animale (nume,data_nasteri,rasa,stapan)…
         values ('',app.NumeanimalEditField.Value,'','',…
        app.DatanasteriiEditField.Value,'','',app.RasaDropDown.Value,…
        '','',app.StapanEditField.Value,'');')
        exec(app.conn,comanda);
end
```

Interfața **Animale – Status** (Fig. 3.30) pune în prim plan animalul de companie. Meniul **Animal** ne permite selectarea unui animal de companie din lista celor înregistrate la clinică. Apoi prin cele trei tab-uri disponibile putem modifica informațiile despre stapan (tab – Stapan, Fig. 3.30), putem adauga data și tipul vaccinului (tab – Vaccin, fig. 3.31) sau data şi tipul medicamentului (tab – Deparazitare, Fig. 3.32) dacă animalului i sa administrat vreun tratament.

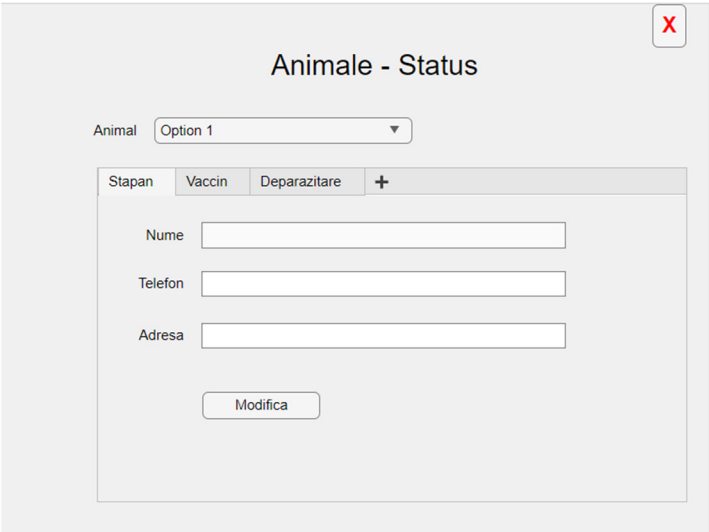


Fig. 3.30. Interfața Status Animal – tab Stapan

Aceasta este interfața care accesează mai multe tabele din baza de date. Interfețele prezentate anterior accesau doar un tabel al bazei de date și permiteau modificarea listelor cu vaccinuri respectiv medicamente. Această interfață are nevoie să acceseze toate tabelele bazei de date și se bazează pe relațiile de legatură dinte tabele.

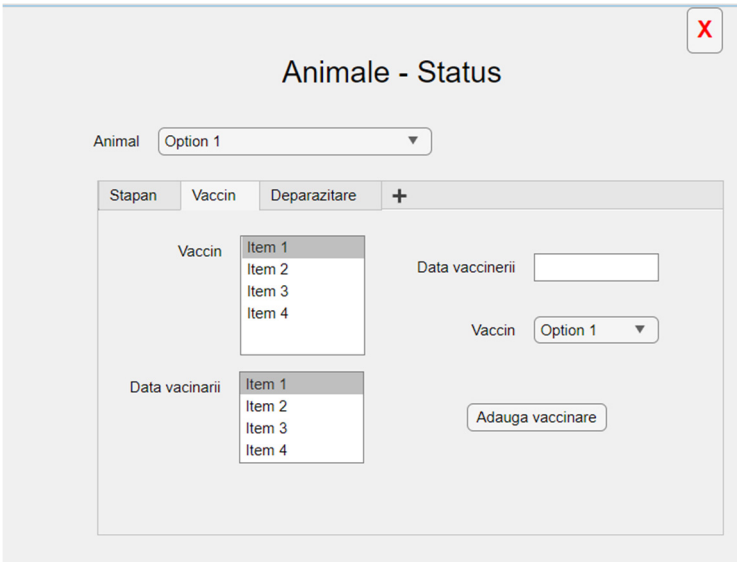


Fig. 3.31. Interfața Status Animal – tab Vaccin

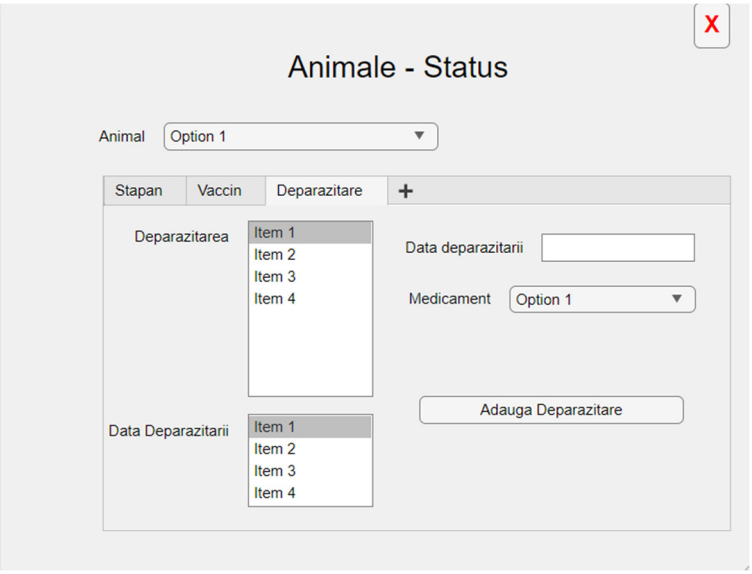


Fig. 3.32. Interfața Status Animal – tab Deparazitare

Codul sursă pentru interfața **Animale - Status** este prezentat în continuare.

```
properties (Access = private)
        conn % Description
end
% Code that executes after component creation
function startupFcn(app)
        app.conn=getappdata(0,'conectare');
        comanda='select nume from animale';
        rezultat=fetch(app.conn,comanda);
        app.AnimalDropDown.Items=rezultat.nume;
        comanda='select vaccin from vaccin';
        rezultat=fetch(app.conn,comanda);
        app.VaccinDropDown.Items=rezultat.vaccin;
        comanda='select medicament from medicament';
        rezultat=fetch(app.conn,comanda);
        app.MedicamentDropDown.Items=rezultat.medicament;
        value = app.AnimalDropDown.Value;
        comanda=strcat('select stapan from animale where nume="',value,'";');
        rezultat=fetch(app.conn,comanda);
        app.NumeEditField.Value=rezultat.stapan{1};
```

```
        comanda=strcat('select * from client where nume='',…
        rezultat.stapan{1},'';');
        rezultat=fetch(app.conn,comanda);
        app.TelefonEditField.Value=num2str(rezultat.telefon);
        app.AdresaEditField.Value=rezultat.adresa{1};
        comanda=strcat('select distinct vaccin from vacinare where nume='',…
        value,'';');
        rezultat=fetch(app.conn,comanda);
        if isempty(rezultat)==1
                app.VaccinListBox.Items={};
                app.DatavacinariiListBox.Items={};
        else
                app.VaccinListBox.Items=rezultat.vaccin;
                app.DatavacinariiListBox.Items={};
        end
        comanda=strcat('select distinct deparazitare from deparazitare…
         where nume='',value,'';');
        rezultat=fetch(app.conn,comanda);
        if isempty(rezultat)==1
                app.DeparazitareaListBox.Items={};
                app.DataDeparazitariiListBox.Items={};
        else
                app.DeparazitareaListBox.Items=rezultat.deparazitare;
                app.DataDeparazitariiListBox.Items={};
        end
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        principala;
        closereq();
end
% Value changed function: AnimalDropDown
function AnimalDropDownValueChanged(app, event)
        value = app.AnimalDropDown.Value;
        comanda=strcat('select stapan from animale where nume='',value,'';');
        rezultat=fetch(app.conn,comanda);
        app.NumeEditField.Value=rezultat.stapan{1};
        comanda=strcat('select * from client where nume='',…
        rezultat.stapan{1},'';');
        rezultat=fetch(app.conn,comanda);
```

```
        app.TelefonEditField.Value=num2str(rezultat.telefon);
        app.AdresaEditField.Value=rezultat.adresa{1};
        comanda=strcat('select distinct vaccin from vacinare…
         where nume='',value,'';');
        rezultat=fetch(app.conn,comanda);
        if isempty(rezultat)==1
                app.VaccinListBox.Items={};
                app.DatavacinariiListBox.Items={};
        else
                app.VaccinListBox.Items=rezultat.vaccin;
                app.DatavacinariiListBox.Items={};
        end
        comanda=strcat('select distinct deparazitare from deparazitare …
        where nume='',value,'';');
        rezultat=fetch(app.conn,comanda);
        if isempty(rezultat)==1;
                app.DeparazitareaListBox.Items={};
                app.DataDeparazitariiListBox.Items={};
        else
                app.DeparazitareaListBox.Items=rezultat.deparazitare;
                app.DataDeparazitariiListBox.Items={};
        end
end
% Value changed function: VaccinListBox
function VaccinListBoxValueChanged(app, event)
        value = app.VaccinListBox.Value;
        comanda=strcat('select data from vacinare where nume='',…
        app.AnimalDropDown.Value,'' and vaccin='',value,'';');
        rezultat=fetch(app.conn,comanda);
        app.DatavacinariiListBox.Items=rezultat.data;
end
% Button pushed function: AdaugavaccinareButton
function AdaugavaccinareButtonPushed(app, event)
        if strcmp(app.DatavaccineriiEditField.Value,'')==1
                data=datestr(now,'yyyy-mm-dd');
        else
                data=app.DatavaccineriiEditField.Value;
        end
        comanda=strcat('insert into vacinare (nume,vaccin,data) values ('',…
        app.AnimalDropDown.Value,'','',app.VaccinDropDown.Value,…
```

```
        '','',data,'');');
        exec(app.conn,comanda);
        value = app.AnimalDropDown.Value;
        comanda=strcat('select distinct vaccin from vacinare…
         where nume='',value,'';');
        rezultat=fetch(app.conn,comanda);
        app.VaccinListBox.Items=rezultat.vaccin;
end
% Button pushed function: AdaugaDeparazitareButton
function AdaugaDeparazitareButtonPushed(app, event)
        if strcmp(app.DatadeparazitariiEditField.Value,'')==1
                data=datestr(now,'yyyy-mm-dd');
        else
                data=app.DatadeparazitariiEditField.Value;
        end
        comanda=strcat('insert into deparazitare (nume,deparazitare,data)…
         values ('',app.AnimalDropDown.Value,'','',…
        app.MedicamentDropDown.Value,'','',data,'');')
        exec(app.conn,comanda);
        value = app.AnimalDropDown.Value;
        comanda=strcat('select distinct deparazitare from deparazitare…
         where nume='',value,'';');
        rezultat=fetch(app.conn,comanda)
        app.DeparazitareaListBox.Items=rezultat.deparazitare;
end
% Value changed function: DeparazitareaListBox
function DeparazitareaListBoxValueChanged(app, event)
        value = app.DeparazitareaListBox.Value;
        comanda=strcat('select data from deparazitare where nume='',…
        app.AnimalDropDown.Value,'' and deparazitare='',value,'';');
        rezultat=fetch(app.conn,comanda);
        app.DataDeparazitariiListBox.Items=rezultat.data;
end
% Button pushed function: ModificaButton
function ModificaButtonPushed(app, event)
        comanda=strcat('update client set telefon='',…
        app.TelefonEditField.Value,'', adresa='',app.AdresaEditField.Value,''
        where nume='',app.NumeEditField.Value,'';')
        exec(app.conn,comanda)
end
```

3.9 Aplicație 9 – Elemente de statistică

Se va realiza o aplicație (Fig. 3.33) care va permite citirea datelor dintr-un fișier extern de tip excel. Pe baza datelor citite se vor calcula estimatorii statistici de bază și de asemenea se vor reprezenta datele sub formă de histogramă

Codul sursă al aplicației este următorul:

```
properties (Access = private)
        date % Description
end
% Button pushed function: FisierButton
function FisierButtonPushed(app, event)
        [fisier cale]=uigetfile('*.xlsx');
        app.date=xlsread(strcat(cale,fisier));
        app.FisierulalesEditField.Value=fisier;
        app.UITable.Data=app.date;
        app.MediaAritmeticaEditField.Value=mean(app.date);
        app.MediaGeometricaEditField.Value=geomean(app.date);
        app.MediaArmonicaEditField.Value=harmmean(app.date);
        app.MedianaEditField.Value=median(app.date);
        app.ModululEditField.Value=mode(app.date);
        app.sigmaEditField.Value=std(app.date);
        histogram(app.UIAxes,app.date);
end
% Button pushed function: XButton
function XButtonPushed(app, event)
        closereq();
end
% Button pushed function: HistogramaButton
function HistogramaButtonPushed(app, event)
        if app.CumulativCheckBox.Value==1
        histogram(app.UIAxes,app.date,app.NrIntervaleEditField.Value,…
        'Normalization','cdf')
        else
        histogram(app.UIAxes,app.date,app.NrIntervaleEditField.Value);
        end
end
```

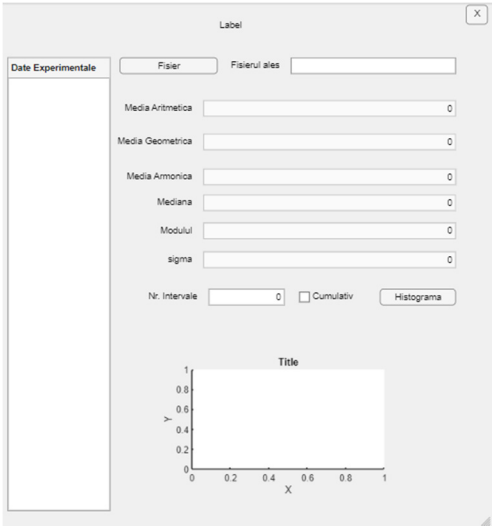


Fig. 3.33. Interfată grafică – Elemente de statistică

Pentru verificarea aplicației s-au încărcat datele din fișierul **TEST1.xlsx**, iar estimatorii statistici calculați sunt prezentați în Fig. 3.34.

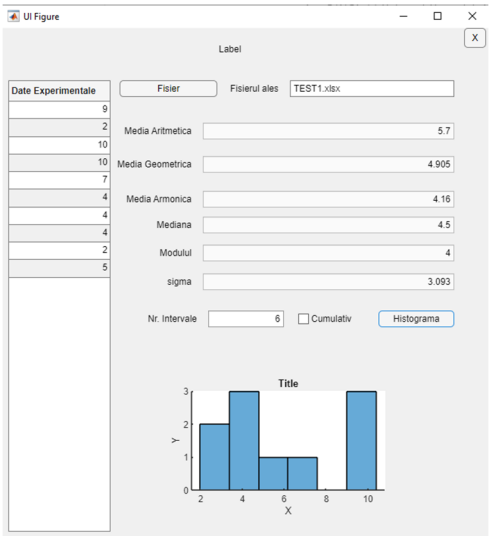


Fig. 3. 34. Rezultate date experimentale

Se observă că s-au calculat toti estimatori statistici. Pentru calculul acestora s-au utilizat funcțiile predefinite în programul Matlab. De asemenea dacă nu se menționează numărul de intervale pentru histogramă atunci acesta va fi cel impliciti.

3.10 Aplicație 10 – Analiza cu element finit

Pentru exemplificare modului în care se poate realiza o analiză cu element finit în aplicația Matlab, s-a realizat modelul CAD 3D (Fig. 3.35) a unui corp ce urmează a fi importat și analizat.

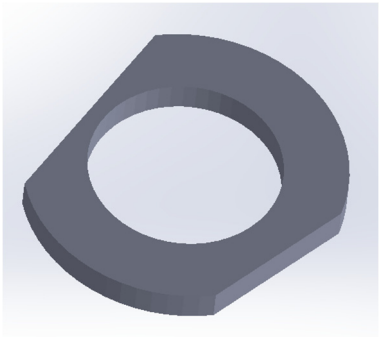


Fig. 3.35. Model CAD

Programul (script-ul) care realizează importul și analiza cu element finit asupra corpului modelat este prezentat în continuare:

```
% import geometry
smodel=createpde('structural','static-solid');
importGeometry(smodel,'test.stl');
figure;
subplot(3,2,1);
pdegplot(smodel,'FaceLabels','on');
title('Model');
mesh=generateMesh(smodel,'Hmax',10);
subplot(3,2,2);
pdeplot3D(smodel);
title('Discretizare');
% preprocesare
E=227E9; %in Pa
nu=0.27;
structuralProperties(smodel,'YoungsModulus',E,'PoissonsRatio',nu);
structuralBC(smodel,'Face',6,'Constraint','fixed');
structuralBoundaryLoad(smodel,'Face',4,'Pressure',500000);
% analiza
Rs=solve(smodel);
subplot(3,2,3);
```

```
pdeplot3D(smodel,'ColorMapData',Rs.VonMisesStress);
title('VonMisesStress');
subplot(3,2,4);
pdeplot3D(smodel,'ColorMapData',Rs.Displacement.ux);
title('Deformatia pe axa X');
subplot(3,2,5);
pdeplot3D(smodel,'ColorMapData',Rs.Displacement.uy);
title('Deformatia pe axa Y');
subplot(3,2,6);
pdeplot3D(smodel,'ColorMapData',Rs.Displacement.uz);
title('Deformatia pe axa Z');
```

În urma rularii acestui script se obțin următoarele rezultate grafice (Fig. 3.36). Variabila **RS** este de tip structură și conține toate rezultatele numerice ale analizei efectuate. Reprezentarea rezultatelor s-a făcut sub formă grafică deoarece este mai ușor de înteles. În cazul acestor reprezentări grafice pot fi formatate proprietățile grafice. Poate fi modificat gradientul de culoare, poate fi modificată paleta de culori, etc.

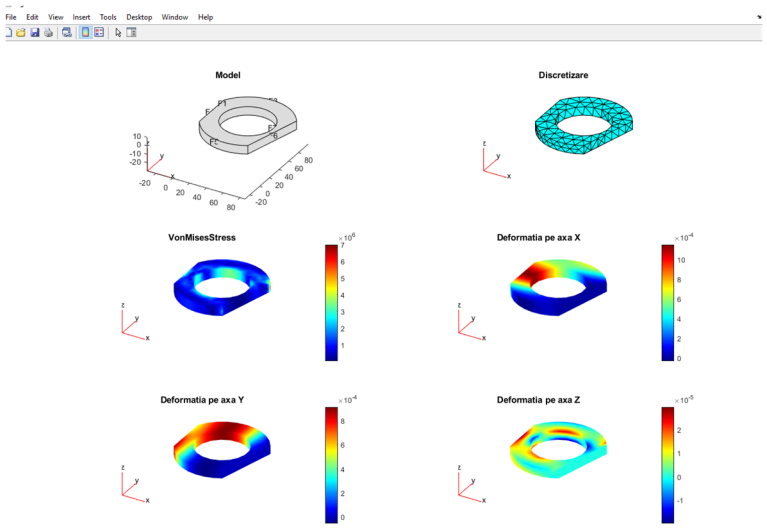


Fig. 3.36. Rezultate analiză cu element finit

De asemenea pentru fiecare reprezentare grafică pot fi utilizate facilitățile programului Matlab care permite rotația, translația, mărirea astfel încât prezentarea rezultatelor analizei cu element finit să fie cât mai sugestiv posibil.

Bibliografie

[1] Arjana Davidescu, *Analiza şi procesarea datelor în Matlab*, Editura Politehnica Timişora, 2003;

[2] https://www.mathworks.com/products/matlab.html

[3] https://www.mysql.com/

[4] https://www.apachefriends.org/

ISBN 978-606-35-0520-1