4. Construirea interfeţelor grafice utilizator – GUI

Mediul de dezvoltare a GUI (GUIDE – graphical user interface development environment) permite construirea rapidă a interfeţelor grafice utilizator pentru un program MATLAB.

GUI conţine două fişiere:

- un fişier pentru reprezentarea grafică cu extensia .fig:
- un fişier pentru aplicaţia propriu-zisă, cu extensia .m.

Fişierul de tip m conţine funcţia principală a interfeţei grafice utilizator şi un număr de subfuncţii. Fişierul se scrie într-un format prestabilit şi se editează pentru a face modificările necesare şi conexiunile dintre diferitele funcţii.

4.1. Dezvoltarea unei interfeţe grafice utilizator

Pentru a lansa mediul de dezvoltare a interfeţei grafice utilizator se tastează în fereastra de comandă: guide. Se va deschide fereastra din fig. 4.1 ce oferă o serie de şabloane posibile pentru GUI, din care se va selecta opţiunea Blank GUI.

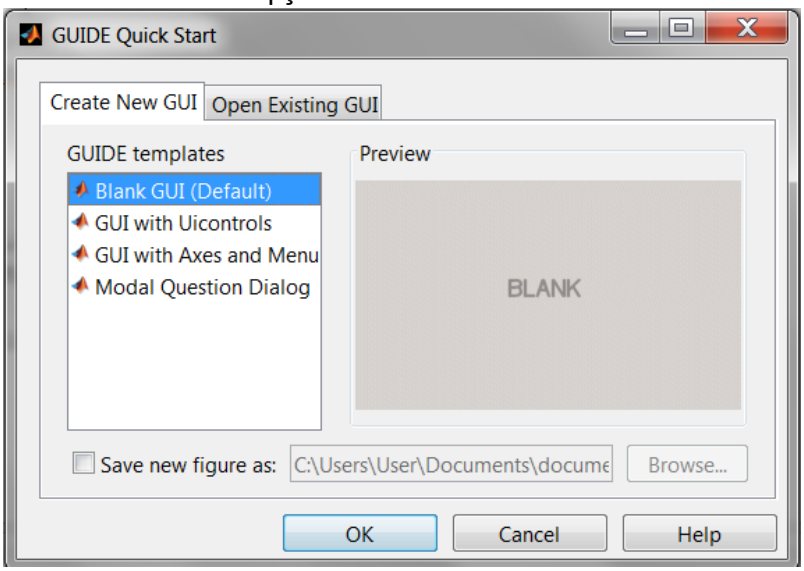


Fig. 4.1 Fereastra de deschidere GUIDE

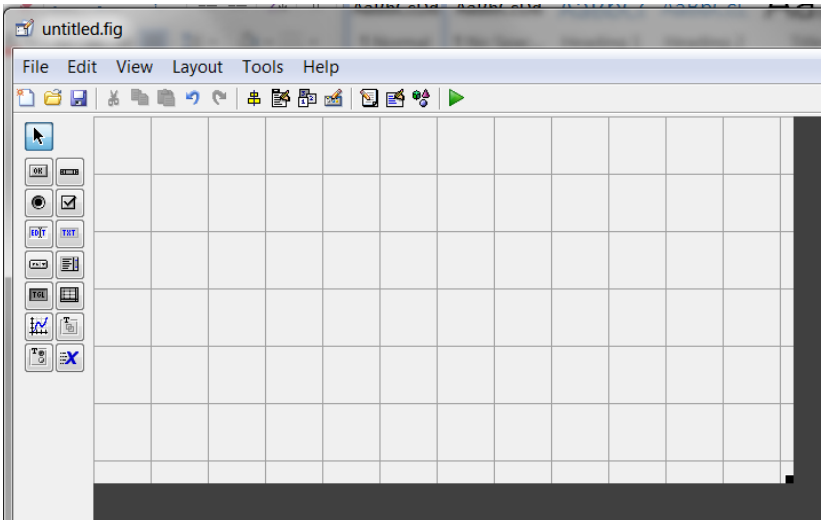


Fig. 4.2 Fereastra GUIDE

Se va deschide fereastra din fig 4.2. Fereastra are trei zone:

- bara de meniu şi pictogramele din partea superioară;
- butoanele pentru selectarea obiectelor din GUI, aflate în partea stângă;

- zona de lucru, în care se construieşte interfaţa grafică propriu-zisă.

Fereastra poate fi redimensionată, ca orice obiect grafic, prin selectarea colţului negru din dreapta jos şi tragerea panoului la dimensiunea dorită. Plasarea controalelor în intefaţă se poate face prin selectarea butonului cu pictograma corespunzătoare şi tragerea elementului în zona dorită. Majoritatea acestor controale sunt instanţe ale clasei uicontrol (user interface controls).

Obiectele care se găsesc în zona din stânga sunt preyentate în fig. 4.3

Buton - Push button	Bară derulantă – Slider
Buton radio – Radio button	Casetă de validare – Checkbox
Casetă text – Edit textbox	Etichetă – Static textbox
Meniu pop-up	Casetă cu listă – listbox
Buton bistabil – Togglebutton	Tabel
Axe	Panou – panel
Grup de butoane	Control ActiveX

Fig. 4.3. Controale

Meniul pop-up oferă o listă de opţiuni pentru ca utilizatorul să poată selecta opţiunea dorită. Caseta cu listă permite parcurgerea opţiunilor existente, făcând una sau mai multe selecţii. Grupul de butoane permite gruparea mai multor butoane de tip radio sau bistabil, pentru a lucra în grup, adică doar unul poate fi selectat la un anumit moment de timp.

Orice obiect din GUI poate fi redimensionat prin selectare şi tragere. Pentru ajustări fine ale poziţiei unui obiect se pot utiliza săgeţile de la tastatură.

După aducerea controalelor dorite în zona de lucru, se salvează interfaţa sub un anumit nume: testGUI sau testTool. Este foarte important ca fişierul corespunzător GUI să conţină în nume sufixul GUI sau Tool, pentru a-l putea distinge de restul fişierelor.

În momentul salvării se generează 2 fişiere:

- testGUI.fig – conţine toate informaţiile dimensionale şi legate de proprietăţile obiectelor introduse în interfaţă;
- testGUI.m – conţine informaţiile necesare pentru construirea şi rularea interfeţei grafice utilizator.

Fişierul cu extensia .fig este un fişier binar, nu poate fi editat decât prin intermediul mediului de dezvoltare GUI. Fişierul cu extensia .m este un fişier MATLAB, ce lansează interfaţa grafică utilizator.

O posibilitate prin care elementele interfeţei interacţionează cu codul este interceptarea unor evenimente, de ex. apăsarea unui buton.

Fie o interfaţă grafică utilizator ce conţine un buton. Se porneşte guide, se aduce un

buton în zona de lucru. În faza iniţială pe buton apare textul “Push Button”. Toate controalele au o serie de proprietăţi, ce pot fi modificate faţă de setarea implicită. Pentru a modifica o anumită proprietate, se execută dublu-click pe controlul dorit şi se va deschide fereastra de proprietăţi a controlului respectiv. Fereastra este organizată pe două coloane, prima conţine numele proprietăţii, iar a doua valoarea acesteia.

Se vor modifica trei proprietăţi ale butonului:

- proprietatea Tag, din pushbutton1 în goButton;
- proprietatea String, din Push Button, în Go!;
- proprietatea FontSize din valoarea 8 (sau altă valoare) în valoarea 14.

Textul ce apare pe buton este determinat de proprietatea String. Valoarea proprietăţii Tag este un şir (nu se acceptă spaţii) ce identifică, în mod unic, obiectul şi constituie numele obiectului. În acest moment, numele butonului este goButton. Această proprietate trebuie aleasă cu atenţie şi asociată cu rolul obiectului în interfaţă.

Este foarte important la construirea interfeţelor grafice utilizator să se atribuie numele controalelor (proprietatea Tag) cât mai reprezentative. Nu sunt premise semne de punctuaţie. Se recomandă formarea numelor prin concatenarea a două părţi:

- prima parte descrie variabila sau cantitatea care este reprezentată sau este data de intrare pentru aplicaţie;
- a doua parte reflect clasa obiectului, de regulă se scrie cu majusculă.

După modificarea proprietăţiilor, se redimensionează fereastra şi se salvează cu numele startGui. Prin tastarea acestui nume în fereastra de comandă se va porni aplicaţia. În fişierul startGui.m apar patru funcţii:

- startGui este funcţia principală, ea construieşte interfaţa propriu-zisă;
- startGui_OpeningFcn – este o funcţie ce se execută la lansare, aici se introduc o serie de iniţializări;
- startGui_OutputFcn – este o funcţie pentru utilizatori experimentaţi, se ignoră la acest moment;
- goButton_Callback – se execută după executare click pe buton.

Funcţiile de tip Callback se asociază cu obiecte particulare din interfaţa grafică utilizator şi se execută când un obiect este activat. Numele funcţiei este generat automat de MATLAB pe baza proprietăţii Tag a obiectului. Pentru ca să se întâmple ceva la acţionarea butonului trebuie să se adauge cod în această funcţie. Se va insera codul următor:

```
disp('Go, go, go!!!');
```

La apăsarea butonului, în fereastra de comandă va apare Go, go, go!!!

În funcţiile de tip callback se introduce cod şi la fiecare eveniment interceptat de interfaţa grafică utilizator, acest cod se execută. Se pot include şi apelări ale altor funcţii, salvate în fişiere separate.

Pentru a exemplifica modul cum se pot prelua informaţii din interfaţa grafică utilizator se va construi o nouă interfaţă, ce conţine: o bară derulantă, o casetă text şi o etichetă (static text). Modificarea proprietăţiilor diferitelor controale introduse în interfaţă este foarte importantă. Informaţii suplimentare legate de semnificaţia şi posibilele valori ale unei proprietăţi se pot obţine prin executarea unui click dreapta pe numele proprietăţii, în fereastra de proprietăţi. Va apare un buton ‘What’s this’, unde, dacă se execută click, apar o serie de explicaţii suplimentare.

Pentru interfaţa curentă se modifică următoarele proprietăţi:

Bara derulantă: Tag – xSlider
Value – 1 (Deoarece pentru unele obiecte proprietatea Value poate fi un vector, ea se setează prin executarea unui click pe pictograma din dreapta proprietăţii Value, în fereastra de proprietăţi.
Max – 20

Caseta text: Tag – xText
String – 1

Etichetă: String – x[m]

Interfaţa trebuie să permită utilizatorului să seteze fizic poziţia lui x la o anumită valoare exprimată în metri, cuprinsă între 0 şi 20.

Pentru a putea comunica cu un obiect GUI, trebuie să fie cunoscută adresa obiectului din memoria calculatorului, codificată de MATLAB printr-un număr real. De asemenea, MATLAB oferă un mecanism facil de obţinere a acestei adrese.

Comunicarea cu controalele din GUI se face cu comenzile get şi set. Cu get se obţin informaţii legate de starea sau proprietăţiile controlului, iar cu set se atribuie valori sau proprietăţi acestuia. Sintaxa comenzii get este:

get(nume_obiect, nume_proprietate)

Nume_obiect este stocat în proprietatea Tag. Ex. get(hSlider, 'Value'), get(hSlider,'Max')

Sintaxa comenzii set este:

set(nume_obiect,proprietate,valoare)

Ex. set(hSlider,'Value',3). Pentru a obţine textul introdus în caseta text se poate utiliza o comandă de forma:

text=get(xText,'String');

Trebuie reţinut faptul că o casetă text conţine informaţie de tip şir de caractere, deci, pentru a prelua o valoare numerică dintr-o casetă text, şirul de caractere trebuie convertit în valoare numerică cu comanda str2num(text). Dacă o apentru fiecarenumită valoare numerică trebuie introdusă într-o casetă text, valoarea respectivătrebuie convertită în şir de caractere:

set(xText, 'String', num2str(valoare_numerica))

În codul asociat interfeţei SliderGUI.m există un număr de funcţii de tip callback, asociat cu fiecare control introdus în interfaţă. De asemenea, există, pentru fiecare control şi funcţii de tip CreateFcn.

Fiecare funcţie callback conţine structura handles transferată ca argument, ceea ce permite accesarea controlului. De ex. handles.xSlider conţine xSlider, handles.xText conţine xText, care, de fapt, este un pointer către obiectul respectiv.

Pentru a corela textul din caseta text cu poziţia cursorului de la bara derulantă, trebuie introdus cod în funcţia callback a barei derulante, care se execute de fiecare data când poyiţia cursorului este modificată. Codul trebuie să realizeze următoarele operaţii:

- să preia noua valoare a culisei, pe care să o reţină într-o variabilă x;
- să convertească în şir această valoare;
- să seteze proprietatea String a casetei în conformitate cu acest şir.

```
Function xSlider_Callback(hObject,eventdata,handles)
%comentarii omise
x=get(handles.xSlider,'Value');
xs=num2str(x);
set(handles.xText,'String',xs);
```

În cazul când se doreşte să se modifice poziţia culisei pe baza valorii introduce în caseta text, codul va trebui introdus în funcţia callback a casetei text. Codul trebuie:

- să preia valoarea din caseta text;
- să convertească şirul de caractere în valoare numerică;
- să atribuie valoarea respectivă proprietăţii Value a barei derulante.

```
function xText_Callback(hObject,eventdata,handles)
%comentarii omise
xs=get(handles.xText,'String');
x=str2num(xs);
set(handles.xSlider,'Value',x);
```

Concluzii

Guide se utilizează pentru construirea GUI. Se generează două fişiere, unul cu extensia .fig, fişier binar ce conţine informaţii legate de aspectul interfeţei şi proprietăţiile controalelor, iar al doilea are extensia .m şi constă dintr-o funcţie principal şi mai multe

subfuncţii, inclusiv de tip callback.

Fiecare control din interfaţa grafică utilizator are o listă de proprietăţi specifice clasei căreia îi aparţine. Proprietăţiile pot fi editate în fereastra de proprietăţi. Proprietatea Tag este foarte importantă deoarece are rolul de identificare a numelui fiecărui obiect particular.

Funcţiile de tip callback se execută la activarea controlului asociat prin acţiuni ale utilizatorului. Numele funcţiei de tip callback se generează automat de mediul de dezvoltare al GUI, pe baza proprietăţii Tag a controlului.

Pointerii către obiecte din interfaţa grafică utilizator sunt versiuni codate ale adresei acestora în memoria calculatorului. Pointerul se se utilizează pentru comunicarea cu obiectul, în principal prin comenziile set şi get. În interiorul unei funcţii de tip callback, pointerii tuturor obiectelor sunt disponibile prin structura handles (aparţinînd clasei struct). GUIDE creează câmpurile structurii handles în concordanţă cu proprietatea Tag a obiectului. De ex. pointerul pentru un obiect având proprietatea Tag xText, va fi handles.xText.

Pentru aflarea valorii unei proprietăţi a unui obiect se utilizează comanda get cu specificarea pointerului către obiect şi a proprietăţii, iar pentru a seta o proprietate la o anumită valoare, se utilizează comanda set, specificînd pointerul către obiect, proprietatea şi valoarea.

4.2. Adaptarea unui program MATLAB într-o interfaţă grafică utilizator

Pentru modelarea comportamentului unui sistem, indiferent de natura lui şi legilor care îl guvernează, sistemul trebuie să fie caracterizat printr-o model matematic bine definit. Modelul va consta dintr-un set de relaţii matematice (ecuaţii algebrice, ecuaţii diferenţiale, ecuaţii matriciale etc.) şi un anumit număr de parametri (ex. mase, concentraţii, tensiuni, constante elastice, viteze iniţiale etc.).

Primul pas este realizarea unei aplicaţii MATLAB care să reflecte comportamentul. Această aplicaţie poate fi un fişier .m sau o funcţie, care pot apela, la rândul lor, alte funcţii definite de utilizator. Modelul matematic trebuie convertit într-un algoritm. Trebuie stabilite: structura datelor (tipul variabilelor utilizate), modul de afişare a rezultatelor sau de comparare cu date experimentale.

Construirea unei GUI va permite utilizatorului să modifice parametri modelului şi să vizualizeze rezultatele într-un mod interactiv, ceea ce facilitează înţelegerea comportamentului.

Pentru a putea dezvolta o GUI, ce modelează comportamentul unui sistem, trebuie parcurse următoarele etape:

- realizarea unui program ce modelează comportamentul sistemului, sub forma unui fişier cu extensia .m;
- planificarea GUI – stabilirea aspectului, a modalităţilor de interacţiune cu utilizatorul pentru modificarea parametrilor, stabilirea modului de afişare a rezultatelor;
- construirea GUI cu GUIDE;
- transformarea programului funcţional în funcţia principală a interfeţei grafice utilizator;
- conectarea funcţiei principale la programul iniţial.

Pentru exemplificare se ia un model foarte simplu, ce constă în evaluarea unei funcţii şi reprezentarea ei grafice. Funcţia reprezintă intensitatea unui câmp electric alternativ ecranat de un strat de dielectric. Se consideră grosimea stratului cuprinsă între 0 şi 10 μm.

Pas 1. Construirea fişierului cu extensia .m, prezentat în continuare

```
%% desenCpEAmortizat
% reprezinta grafic un camp electric sinusoidal pentru grosimea
% dielectricului intre 0 si 10 micrometri
```

```
%% date intrare
a=3; % grosimea dielectric
E0=12; %Intensitatea campului la x=0 [V/micrometri]
lambda=1; % Lungimea de unda
```

```
xmin=0;
xmax=10;
n=200;
```

```
%% initializare variabile
x=linspace(xmin,xmax,n);
E=zeros(1,n);
```

```
%% calcule
k=2*pi/lambda;
E=E0*cos(k*x).*exp(-x/a);
```

```
%% reprezentare
plot(x,E)
axis([xmin xmax -E0 E0])
xlabel('x(micrometri');
ylabel('E(V/micrometri)');
grid on
```

Pas 2. Planificarea GUI

În acest pas trebuie stabilite datele de intrare care se vizualizează în GUI. În aplicaţia construită există şase date: a, E0, lambda, xmin, xmax, n. Datele trebuie analizate cu atenţie pentru că existenţa unui număr mare de parametri poate conduce la o interfaţă mult prea confuză pentru utilizator. În cazul de faţă, datele pot fi grupate în două categorii, unele ce au semnificaţie fizică (a, E0, lambda) şi a doua categorie, date de programare (xmin, xmax, n). Se optează pentru prezentarea în interfaţă doar a grosimii dielectricului, a, în sensul că utilizatorul să aibă posibilitatea modificării cu uşurinţă a acestei grosimi şi să vizualizeze rezultatul. Acest lucru poate fi asigurat printr-o bară derulantă, o casetă text pentru afişarea grosimii stratului şi un sistem de axe pentru reprezentarea grafică.

Se recomandă ca în interfeţele grafice utilizator să apară numele aplicaţiei, deci se va introduce şi o etichetă, iar numele fişierului va fi CampEAmortizatGUI.

Tot în această etapă se recomandă să se stabilească proprietatea Tag a fiecărui obiect. Este recomandabilă chiar schiţarea interfeţei grafice cu notarea proprietăţilor controalelor. De asemenea, se stabilesc şi alte proprietăţi legate de domeniul de valori pentru reprezentarea grafică, proprietatea Min şi Max pentru bara derulantă etc.

Pas 3. Utilizarea GUIDE pentru construirea GUI

Se construieşte interfaţa în varianta prezentată în fig. 4.4.

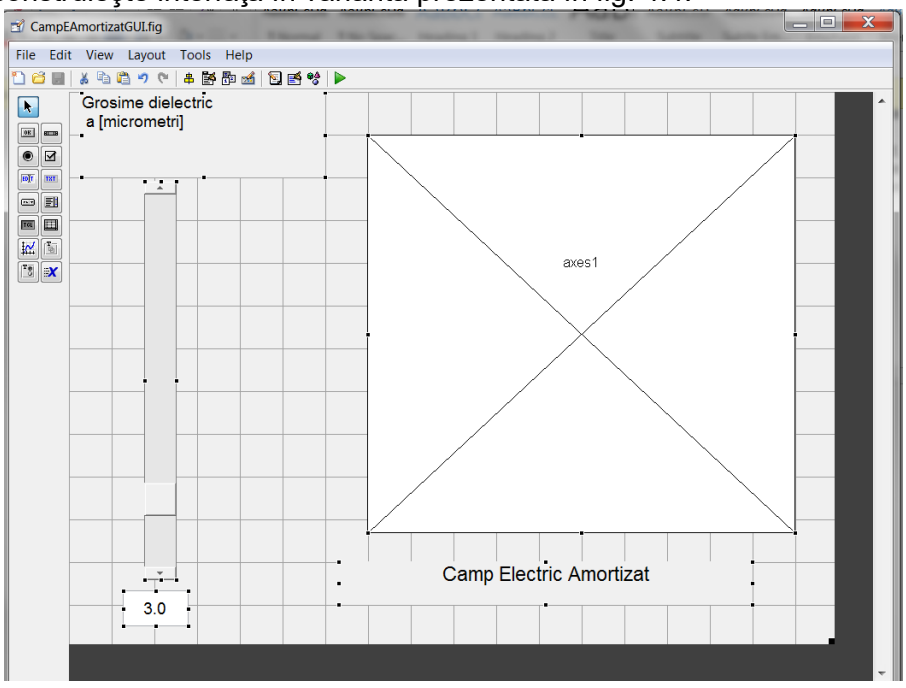


Fig. 4.4 Interfaţa în GUIDE

Proprietăţile controalelor se stabilesc:
Eticheta pentru bara derulantă: FontSize =12
String = Grosime dielectric
a (micrometri)
Bara derulantă: Tag = grosSlider
Max = 20
Value = 3
Caseta text: Tag = grosText
FontSize = 12
String = 3
Sistemul de axe: Tag = plotAxes
Eticheta pentru titlu: FontSize = 14
String = Camp Electric Amortizat

Se observă că se modifică proprietatea Tag doar la acele elemente care interacţionează cu codul aplicaţiei. Etichetele sunt lăsate cu proprietatea generată implicit de GUIDE.

În încheierea etapei se salvează aplicaţia, obţinând fişierele CampEAmortizatGUI.m şi CampEAmortizatGUI.fig.

Pas 4. Modificarea programului

Se încarcă programul (desenCampEAmortizat.m) în editor şi se salvează sub aceeaşi denumire, dar se adaugă sufixul F, de la funcţie. Se va modifica prima linie, transformând fişierul în funcţie, având ca argument structura handles.

Prima linie devine:

```
function desenCpEAmortizatF(handles)
```

Instrucţiunile de atribuire, care se preiau din interfaţă (în cazul de faţă, grosimea a) se înlocuiesc cu comenzi get. Instrucţiunea a = 3 devine:

```
a=get(handles.grosSlider,'Value');
```

În mod implicit, instrucţiunile pentru reprezentare grafică au ca efect desenarea în figura curentă. Pentru a fi siguri de faptul că desenarea se face în zona prevăzută în interfaţă, trebuie adăugat ca argument suplimentar la aceste instrucţiuni pointerul către controlul respectiv, în cazul de faţă handles.plotAxes. Pentru a scrie mai puţin, se poate defini o variabilă hax = handles.plotAxes şi lucra cu ea. Secvenţa de desenare devine:

```
hax=handles.plotAxes;
plot(hax,x,E)
axis(hax,[xmin xmax -E0 E0])
xlabel(hax,'x(micrometri');
ylabel(hax,'E(V/micrometri)');
grid (hax,'on');
```

Este bine să se lucreze în acest mod, deoarece sunt multe situaţii când într-o interfaţă apar mai multe zone de desenare, iar reprezentările trebuie direcţionate corect.

Programul cu modificări este prezentat în continuare:

```
function desenCpEAmortizatF(handles)
% reprezinta grafic un camp electric sinusoidal pentru grosimea
% dielectricului intre 0 si 10 micrometri
%% date intrare
a=get(handles.grosSlider,'Value'); % grosimea dielectric
E0=12; %Intensitatea campului la x=0 [V/micrometri]
lambda=1; % Lungimea de unda
xmin=0;
xmax=10;
n=200;
%% initializare variabile
```

```
x=linspace(xmin,xmax,n);
E=zeros(1,n);
%% calcule
k=2*pi/lambda;
E=E0*cos(k*x).*exp(-x/a);
%% reprezentare
hax=handles.plotAxes;
plot(hax,x,E)
axis(hax,[xmin xmax -E0 E0])
xlabel(hax,'x(micrometri');
ylabel(hax,'E(V/micrometri)');
grid (hax,'on');
```

Mai trebuie rezolvate următoarele aspecte:

- sincronizarea culisei cu caseta text şi vice versa;
- desenarea funcţiei la fiecare modificare a culisei;
- rularea modelului iniţial (a = 3) la prima rulare

Fişierul m asociat interfeţei grafice utilizator constă din următoarele şapte funcţii:

- function CampEAmortizatGUI
- function CampEAmortizatGUI_OpeningFcn – se modifică
- function CampEAmortizatGUI_OutputFcn
- function grosSlider_Callback – se modifică
- function grosSlider_CreateFcn
- function grosText_Callback – se modifică
- function grosText_CreateFcn.

Ori de câte ori se modifică poziţia culisei trebuie să se modifice corespunzător valoarea din caseta text şi să se facă reprezentarea grafică corespunzătoare prin apelarea funcţiei desenCampEAmortizatF(handles). Se va modifica funcţia grosSlider_Callback:

```
a=get(handles.grosSlider,'Value');
astr=num2str(a);
set(handles.grosText,'String',astr);
desenCampEAmortizatF(handles);
```

Pentru a se face actualizarea după ce utilizatorul a introdus o valoare în caseta text, trebuie să se modifice corespunzător poziţia culisei şi să se facă reprezentarea grafică. Funcţia grosText_Callback se va modifica:

```
astr=get(handles.grosText,'String');
a=str2double(astr);
set(handles.grosSlider,'Value',a);
desenCampEAmortizatF(handles)
```

Pentru ca în momentul lansării interfeţei să apară o reprezentare grafică este necesar să se modifice funcţia CampEAmortizatGUI_OpeningFcn prin introducerea instrucţiunii:

```
desenCampEAmortizatF(handles)
```

Forma finală a fișierului m este preyentată în continuare. Liniile marcate îngroșat sunt cele care sunt introduse pentru a realiya funcționarea interfeței.

```
function varargout = CampEAmortizatGUI(varargin)
function CampEAmortizatGUI_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes CampEAmortizatGUI wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);
desenCampEAmortizatF(handles);

% --- Outputs from this function are returned to the command line.
function varargout = CampEAmortizatGUI_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;


function grosSlider_Callback(hObject, eventdata, handles)
a=get(handles.grosSlider,'Value');
astr=num2str(a);
set(handles.grosText,'String',astr);
desenCampEAmortizatF(handles);

% --- Executes during object creation, after setting all properties.
function grosSlider_CreateFcn(hObject, eventdata, handles)

function grosText_Callback(hObject, eventdata, handles)
astr=get(handles.grosText,'String');
a=str2double(astr);
set(handles.grosSlider,'Value',a);
desenCampEAmortizatF(handles)

% --- Executes during object creation, after setting all properties.
function grosText_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

O altă interfață grafică utilizator, prezentată în continuare, modelează lansarea cu viteza v0 a unei mingi de ping-pong, sub un anumit unghi cu orizontala. Se ia în considerare forța de frecare cu aerul pentru a stabili traiectoria, viteza și accelerația mingii, respectiv evoluția energiei cinetice, potențiale și totale. Dintre parametri modelului, masa, diametrul, coeficientul de rezistență aerodinamică, înălțimea și viteza de lansare, respectiv durata de timp pentru care se face modelarea se introduc prin intermediul unor casete text. Unghiul de lansare (măsurat față de orizontală) se poate selecta dintr-un grup de butoane, care permit alegerea unei valori dintre: 0, 15, 30, 45, 60, 75, 90. Într-un sistem de se se reprezintă traiectoria mingii, iar în al doilea sistem de axe se reprezintă energiile. Actualizarea celor două reprezentări grafice se face cu ajutorul unui buton.

În prima etapă trebuie scris fișierul care face modelarea procesului. Forțele ce acționează asupra mingii sunt greutatea și forța de frecare cu aerul:

$$G = mg$$

$$F_a = -\frac{1}{2} C_d A \rho v^2$$

Deci, pentru cele două axe, se pot considera ecuațiile:

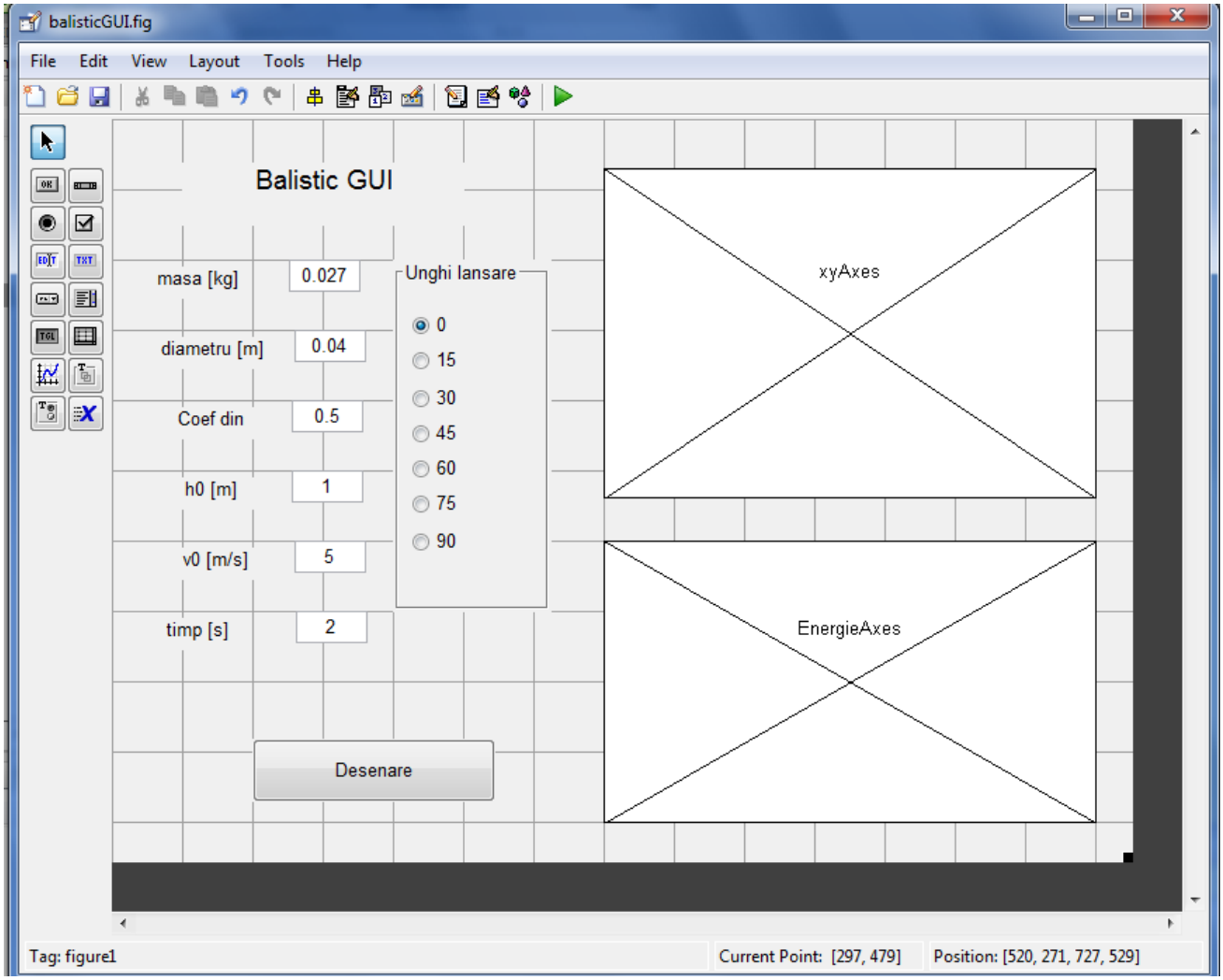
$$\begin{cases} ma_x = kv_x^2 \\ ma_y = -mg - kv_y^2 \end{cases}$$

Plecând de la aceste ecuații se determină expresiile pentru viteze, accelerații și spațiu, considerând accelerațiile inițiale 0, iar vitezele inițiale:

$$\begin{cases} v_{0x} = v_0 cos\alpha \\ v_{0y} = v_0 sin\alpha \end{cases}$$

```
%% balistic.m
% modeleaza lansarea unei mingi de greutate m cu viteza vo si unghi alfa
% sub actiunea greutatii si fortei aerodinamice
%% date intrare
```

```
m=.027; % masa [kg]
d=.04; %diametru [m]
Cd=0.5; %coeficient dinamic pt sfera
ro=1.2; % densitate aer kg/m^3
h0=1; % inaltime de lansare[m]
v0=5; % viteza lansare m/s
alfa=10;% unghi lansare grade
T=2; %timp analiza
deltat=.01; % increment
g=9.81; % acceleratie gravitationala
%% initializare variabile
n=T/deltat;% nr pasi
v=zeros(1,n);
vx=v;
vy=v;
x=v;
y=v;
ax=v;
ay=v;
%% calcule
vx(1)=v0*cosd(alfa);
vy(1)=v0*sind(alfa);
y(1)=h0;
k=-1/2*Cd*ro*pi*d^2/4;
for i=2:n
    vx(i)=vx(i-1)+ax(i-1)*deltat;
    vy(i)=vy(i-1)+ay(i-1)*deltat;
    ax(i)=k*vx(i)/m;
    ay(i)=-g-k/m*vy(i);
    x(i)=x(i-1)+1/2*(vx(i-1)+vx(i))*deltat+1/4*(ax(i-1)+ax(i))*deltat^2;
    y(i)=y(i-1)+1/2*(vy(i-1)+vy(i))*deltat+1/4*(ay(i-1)+ay(i))*deltat^2;
    if y(i)<=0
        vy(i)=-vy(i);
    end
end
v=sqrt(vx.^2+vy.^2);
plot(x,y)
Ec=m*v.^2/2;
Ep=m*g*y;
E=Ec+Ep;
figure,plot(1:n,Ec,1:n,Ep,1:n,E)
```



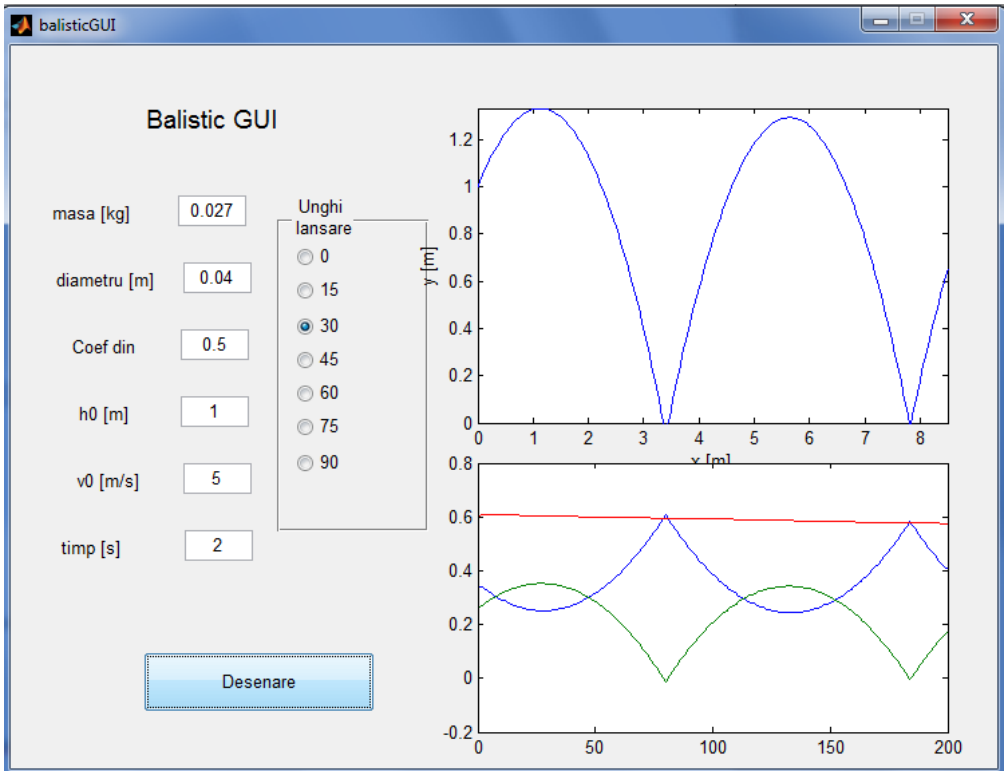
După stabilirea configurației GUI, se modifică corespunzător proprietățiile controalelor. Fișierul balistic.m se salvează sub formă de funcție. Fișierul este de forma:

```
function balisticf(handles)
% modeleaza lansarea unei mingi de greutate m cu viteza vo si unghi alfa
% sub actiunea greutatii si fortei aerodinamice
%% date intrare
%m=.027; % masa [kg]
ms=get(handles.masaText,'String');
m=str2double(ms);
%d=.04; %diametru [m]
ds=get(handles.dText,'String');
d=str2double(ds);
%Cd=0.5; %coeficient dinamic pt sfera
Cds=get(handles.CdText,'String');
Cd=str2double(Cds);
ro=1.2; % densitate aer kg/m^3
%h0=1; % inaltime de lansare[m]
h0s=get(handles.h0Text,'String');
h0=str2double(h0s);
%v0=5; % viteza lansare m/s
v0s=get(handles.v0Text,'String');
v0=str2double(v0s);
%alfa=10;% unghi lansare grade
hSelectedObject=get(handles.unghiButtongroup,'SelectedObject');
alfas=get(hSelectedObject,'String');
```

```
alfa=str2double(alfas);
%T=2; %timp analiza
Ts=get(handles.tText,'String');
T=str2double(Ts);
deltat=.01; % increment
g=9.81; % acceleratie gravitationala
%% initializare variabile
n=T/deltat;% nr pasi
v=zeros(1,n);
vx=v;
vy=v;
x=v;
y=v;
ax=v;
ay=v;
%% calcule
vx(1)=v0*cosd(alfa);
vy(1)=v0*sind(alfa);
y(1)=h0;
k=-1/2*Cd*ro*pi*d^2/4;
for i=2:n
    vx(i)=vx(i-1)+ax(i-1)*deltat;
    vy(i)=vy(i-1)+ay(i-1)*deltat;
    ax(i)=k*vx(i)/m;
    ay(i)=-g-k/m*vy(i);
    x(i)=x(i-1)+1/2*(vx(i-1)+vx(i))*deltat+1/4*(ax(i-1)+ax(i))*deltat^2;
    y(i)=y(i-1)+1/2*(vy(i-1)+vy(i))*deltat+1/4*(ay(i-1)+ay(i))*deltat^2;
    if y(i)<=0
        vy(i)=-vy(i);
    end
end
v=sqrt(vx.^2+vy.^2);
ymax=max(y);
xmax=max(x);
hax1=handles.xyAxes;
plot(hax1,x,y)
axis(hax1,[0,xmax,0,ymax]);
xlabel(hax1,'x [m]')
ylabel(hax1,'y [m]')
Ec=m*v.^2/2;
Ep=m*g*y;
E=Ec+Ep;
hax2=handles.EnergieAxes;

plot(hax2,1:n,Ec,1:n,Ep,1:n,E)
```

Instrucțiunile de atribuire ce trebuie modificate sunt transformate în comentariu, fiind urmate de noile atribuiri.



În multe situații este util să existe mai multe sisteme de axe pentru reprezentări grafice. În astfel de situații este important ca fiecare reprezentare grafică să fie trimisă în zona grafică corespunzătoare. S-a arătat că mai multe comenzi grafice pot fi direcționate la un anumit sistem de axe incluzând ca prim argument handles.NumeAxe. Acest lucru este valabil pentru plot, plot3, xlabel, ylabel, grid, axis, title, box, rotate3D, legend, area, bar, pie, hist, surf.

Mai există o serie de comenzi grafice, ce adaugă la figura curentă un anumit obiect grafic: rectangle, line, patch, surface, text, hgtransform, image, Aceste comenzi nu acceptă ca prim argument handles.NumeAxe. In astfel de situații trebuie specificată proprietatea Parent la handles.NumeAxe.

Ex. Pentru a construi un pătrat de culoare verde:

```
patch([0,1,1,0], [0,0,1,1],'g','Parent',handles.animAxe)
```

sau pentru a desena o linie, de culoare neagră și grosime 2

```
hline=line([0,1.4],[0,1.4]);
set(hline,'Parent',handles.animAxe,'LineWidth',2,'Color','k');
```

În primul caz specificarea proprietății Parent se face simultan cu creearea obiectului, iar în al doilea caz se face după ce obiectul a fost creat.

Dacă o GUI efectuează calcule scurte este normal ca la orice modificare să răspundă, dar în cazul când calculele sunt de durată sau implică animația, se recomandă ca simularea să se producă doar la solicitarea utilizatorului (prin apăsarea unui buton).

Probleme

1. Să se construiască o GUI pentru reprezentarea funcției

$$f(x) = e^{-x/a} cos\left(2\pi \frac{x}{\lambda}\right)$$

pentru x ϵ[-5, 5], a ϵ[0.05, 50] și λ ϵ[0.05, 20]. Pentru modificarea parametrului a și λ se folosesc bare derulante și casete text pentru editarea valorii.

2. Să se construiască o interfață grafică utilizator pentru reprezentarea funcției:

$$f(x) = \frac{1}{1 + e^{-\frac{y-y_0}{a}}}$$

y0 și a se vor modifica prin intermediul unei bare derulante. Se vor stabili domeniile cele mai reprezentative de valori.

3. Să se construiască o aplicație pentru reprezentarea curbei definite parametric:

$$\begin{cases} x = \cos(a\omega t) \\ y = \sin(b\omega t + \varphi \end{cases}$$

unde ω=2π/T. t $\epsilon[0, T]$, a,b ϵ[0,10], φ ϵ[0, 2π]. Aplicația se va transforma în GUI

4. Construiți o GUI care să contorizeze apăsările pe un buton. Interfața va conține o casetă text, unde se afișează contorul, un buton la a cărui apăsare se incrementează contorul, un buton pentru resetarea contorului
5. urm.///
6. ……..

4.3. Componente GUI

În continuare se prezintă o succintă trecere în revistă a obiectelor de bază din GUI și a modului uzual de utilizare. Explicații extinse se pot găsi în documentația MATLAB.

4.3.1. Buton

Descriere: principala acțiune asociată cu butonul este apăsarea și eliberarea acestuia pentru a iniția o acțiune prin apelarea funcției de tip callback asociată.

Clasa: uicontrol, Style = pushbutton

Proprietatea de bază: String

Prezintă interes și alte proprietăți asociate cu aspectul textului de pe buton: FontName, FontWeight (normal și bold, la variantele mai vechi există și opțiunea tight și demi), FontAngle (normai și italic, la variantele mai vechi există si oblique).

Proprietatea ForegroundColor determină culoarea textului (proprietatea String), iar proprietatea BackgroundColor setează culoarea fundalului. Ambele culori se pot modifica prin utilizarea instrumentului de alegere a culorii sau se pot modifica dinamic cu comanda set (fie printr-un vector de 3 elemente, culorile RGB, cu valori între 0 și 1; ex. [0.8,0.8,1] – albastru deschis). Nu se recomandă exces de culoare în interfețele grafice utilizator, culoarea gri, implicită, conferă mai mult profesionalism interfeței grafice.

4.3.2. Bara derulantă

Descriere: utilizatorul mută și eliberează culisa pentru a specifica o valoare între valoarea proprietății Min și Max.

Clasa: uicontrol, Style = slider

Proprietăți de bază: Value, Min, Max

Pentru a extrage valoare de la bara derulantă se poate utiliza codul:

v_ini=get(handles.viniSlider,'Value')

Modificarea poziției culisei, deci a proprietății Value, poate fi făcută prin executarea unui click pe săgețile obiectului, modificarea se face cu increment mic, sau cu valori mai mari, prin apăsarea pe bara propriu-zisă. Mărimea incrementului cu care se face modificarea este determinate de proprietatea SliderStep, care este un vector cu două elemente, [MinorStepSize, MajorStepSize]. Valoarea implicită este [0.01, 0.1] , Valoarea este exprimată relativ, înseamnă 0.01(Max-Min) și 0.1(Max-Min).

4.3.3. Caseta de validare

Descriere: se utilizează pentru bifarea sau debifarea unei casete cu etichetă. De regulă, se asociază cu o decizie binară, fiind utilizată la setarea unei valori logice.

Clasa: uicontrol, Style = checkbox

Proprietatea de bază: Value, String.

Proprietatea Value este logică – true sau false (1 sau 0), valoarea true se asociază cu bifarea casetei. Proprietatea String specific eticheta din dreptul casetei.

Ex.

```
% extragerea valorii logice a casetei de validare gridCheckbox
caroiaj=get(handles.gridCheckbox,'Value');
……
if caroiaj
    grid(handles.plotAxes,'on');
end
```

4.3.4. Caseta text

Descriere: utilizatorul poate introduce text în casetă prin tastare. Textul se stochează în proprietatea String. Dacă se dorește extragerea unei valori numerice, valoarea extrasă, care este de tip șir, trebuie convertită ăn valoare numerică.

Clasa: uicontrol, Style = edit

Proprietatea de bază: String

Ex. extragerea unei valori numerice din caseta text

```
kstr=get(handles.gText,'String');
k=str2double(kstr);
```

Proprietățiile menționate la buton, legate de aspectul fontului sunt valabile și în cazul casetei text.

Pentru a obține o casetă text multilinie se setează proprietățiile Min și Max (0 și 1, în varaiantă implicită) astfel încât Max –Min > 1. Aceste casete au scroll vertical.

4.3.5. Eticheta

Descriere: se utilizează pentru a afișa un text ce nu poate fi modificat de utilizator.

Clasa: uicontrol, Style = text

Proprietatea de bază: String

Ex.

```
% afisarea valorii unei variabile Etot cu text asociat
estring=['Energia totala este ',num2str(Etot).'Joules'];
set(handles.eText,'String',estring);
```

Etichetele se utilizează frecvent pentru afișarea unor explicații pentru utilizatori. În aceste situații, proprietatea Tag nu trebuie modificată din varianta implicită. În exemplul precedent, eticheta se utilizează pentru afișarea unui rezultat. Proprietățiile legate de aspectul fontului sunt similar cu cele de la buton sau caseta text.

Se pot utiliza fonturi de tip Symbol, putând defini litere grecești (FontName se setează Symbol), iar alinierea textului se face cu proprietatea HorizontalAlignment

4.3.6. Caseta Combo sau meniul popup

Descriere: Caseta Combo este un meniu popup care prezintă utilizatorului o listă de elemente, din care se poate face o singură selecție. Selecția curentă este reprezentată se șirul afișat de meniu în caseta superioară.

Clasa: uicontrol, Style = popupmenu

Proprietăți de bază: String, Value.

Proprietatea String reprezintă conținutul meniului, lista propriu-zisă. Lista se introduce în faza de construcție a interfeței, prin introducerea pe linii separate a fiecărui element.

Cu comanda get se poate extrage proprietatea Value (care este o valoare întreagă), ce corespunde numărului de ordine al elementului selectat.

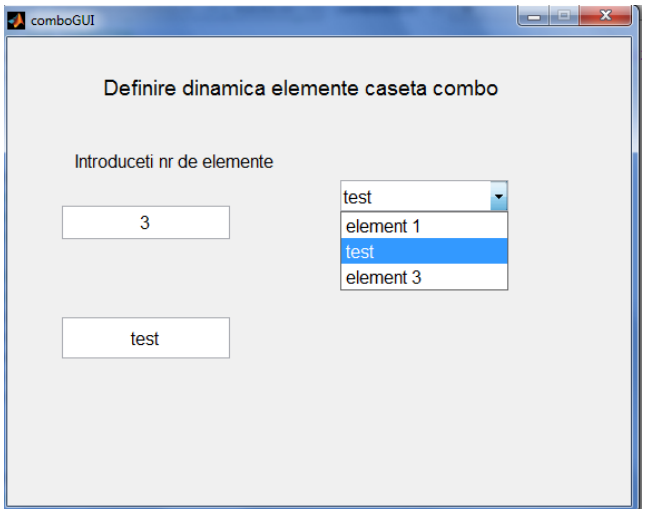
Ex. Extragerea elementului selectat pentru o casetă ce conține opțiuni de reprezentare grafică: spațiu (1), viteză (2), accelerație (3).

tPlot=get(handles.plotPopup,'Value');

Proprietatea String poate fi specificată și prin cod, ea reprezentând o matrice complexă, ce conține șiruri.

set(handles.plotPopup,'String',{'spatiu','viteza', 'acceleratie'});

Posibilitatea de modificare dinamică a proprietății String permite o utilizare foarte flexibilă a meniului. De ex. utilizatorul poate introduce într-o casetă text un număr, pe baza căruia se generează proprietatea String a casetei, fiecare element din listă fiind preluat dintr-o altă casetă text.



Fie interfața grafică utilizator ce conține o casetă text pentru introducerea numărului de elemente al listei, o casetă text pentru introducerea elementului din listă, o casetă combo și eventual un buton, dacă se dorește să se facă atribuirea elementului după acționarea butonului. La introducerea unei valori numerice în caseta text, nText, se generează automat o listă cu un număr corespunzător de elemente, având denumirea generică element 1, element 2 etc. Prin selectarea elementelor meniul popup, acestea sunt transferate casetei text elemText, unde utilizatorul introduce noul element. La tastarea Enter noul element se introduce în listă.

Codul pentru inițializarea interfeței, respectiv din funcția callback a casetei nText este:

```
n=str2double(get(handles.nText,'String'));
for k=1:n
    el_pop{k}=['element ',num2str(k)];
end
set(handles.MateriiPopup,'String',el_pop);
set(handles.MateriiPopup,'Value',1);
set(handles.elemText,'String',el_pop(1));
```

Codul din funcția callback a casetei elemText este:

```
index=get(handles.MateriiPopup,'Value');
sir=get(handles.elemText,'String');
elemente=get(handles.MateriiPopup,'String');
elemente{index}=sir;
set(handles.MateriiPopup,'String',elemente);
```

Pentru a realiza preluarea elementului selectat din listă și transferarea acestuia în caseta elemText, se utilizează codul (în funcția callback a casetei combo:

```
v=get(handles.MateriiPopup,'Value');
```

```
elemente=get(handles.MateriiPopup,'String');
sir=elemente(v);
set(handles.elemText,'String',sir);
```

4.3.7. Caseta cu listă

Descriere: oferă o listă de opțiuni din care utilizatorul poate face una sau mai multe selecții, respectiv poate să nu facă nicio selecție.

Clasa: uicontrol, Style = listbox

Proprietăți de bază: String, Value, Min, Max.

În cazul când Min = 0 și Max = 1 (valorile implicite), caseta cu listă se comport ca un meniu popup, dar lista poate fi vizualizată prin derulare. Proprietatea Value conține o valoare întreagă reprezentând indexul elementului selectat.

Dacă Max = 2 (sau orice alt număr întreg), utilizatorul poate selecta mai multe elemente. Prima selecție se face prin click, iar următoarele prin tastare Ctrl+click. Tastarea a doua oară Ctrl+click determină deselectarea elementului. Un grup de elemente successive poate fi selectat prin clck pe primul element și Shift+click pe ultimul.

Proprietatea Value conține un vector de numere întregi, ce reprezintă indicia elementelor selectate. Dacă nu este făcută nicio selecție, vectorul este de lungime 0.

Proprietatea String conține o matrice complexă, în care sunt stocate șirurile ce reprezintă elementele listei.

În urnătorul exemplu funcția de tip callback a unei casete cu listă preia proprietatea Value (vectorul cu indicia elementelor selectate), se citesc elementele din listă cu proprietatea String și se construiește un nou vector cu elementele selectate, care sunt afișate într-o casetă text.

```
      function materiiListbox_Callback(hObject, eventdata, handles)
      materiiSel=get(handles.materiiListbox,'Value'); % indicii
selectati
      materiiSir=get(handles.materiiListbox, 'String');  % lista
completa
      n=length(materiiSel);  % numar de selectii
      if n>0
            comentariu=['Materiile alese:
',materiiSir{materiiSel(1)}];
            for i=2:n

      comentariu=[comentariu,',',materiiSir{materiiSel(i)}];
            end
            comentariu=[comentariu,'.'];
      else
            comentariu='Nu s-a ales nicio materie.';
      end
      set(handles.materiiText,'String', comentariu);
```

4.3.8. Grup de butoane, buton on/off și buton radio

Descriere: un grup de butoane conține mai multe tipuri de butoane, fie de tip on/off sau butoane radio și permite selectarea unui singur element din grup.

Clasa: uibuttongroup, uicontrol, Style = togglebutton, uicontrol, Style = radiobutton.

Proprietatea de bază: SelectedObject.

Un grup de butoane poate fi utilizat pentru a afișa un set de opțiuni mutual exclusive, evidențiate fie prin butoane on/off sau radio. Funcția este foarte asemănătoare cu a unui meniu popup. Obiectul grup de butoane controlează selecția unică.

Pentru a determina care este selecția, trebuie preluată (get) valoarea proprietății SelectedObject. Ulterior cu proprietatea String sau Tag a obiectului selectat se poate determina ce acțiune trebuie întreprinsă.

Funcțiile callback pentru butoanele individuale sunt dezactivate. Pentru a putea declanșa o acțiune la modificarea selecției trebuie inserată funcția SelectionChangeFcn, care nu se face în mod implicit. Pentru a realiza acest lucru, în GUIDE trebuie selectat grupul de butoane și din bara de meniu se selectează View/View Callback/SelectionChangeFcn. Efectul acestei comenzi este adăugarea la fișierul aplicației a șablonului corespunzător acestei funcții. În interiorul acestei funcții, obținerea valorii proprietății SelectedObject va conduce la posibilitatea manevrării noii selecții. Uneori este avantajos să se obțină proprietatea Tag a acestui buton.

În următorul exemplu se construiește o GUI ce conține un grup de 6 butoane și o casetă text în care se evidențiază selecția făcută de utilizator. Codul corespunzător (aflat în funcția `metodaButtongroup_SelectionChangeFcn`) este prezentat în continuare:

```
hSelectedObj=get(handles.metodaButtongroup,'SelectedObject');
selectedObjTag=get(hSelectedObj,'Tag');
switch selectedObjTag
    case 'MinTogglebutton'
        set(handles.metodaText,'String','Se va selecta valoarea minima');
    case 'MaxTogglebutton'
        set(handles.metodaText,'String','Se va selecta valoarea maxima');
    case 'MedieTogglebutton'
        set(handles.metodaText,'String','Se va calcula media');
    case 'NetezireTogglebutton'
        set(handles.metodaText,'String','Datele se vor netezi');
    case 'DerivareTogglebutton'
        set(handles.metodaText,'String','Datele se vor deriva');
    case 'IntegrareTogglebutton'
        set(handles.metodaText,'String','Datele se vor integra');
end
```

4.3.9. Obiectul panou (panel)

Descriere: Panoul asociază vizual un grup de obiecte prin plasarea unui dreptunghi în jurul lor.

Clasa: uipanel

Proprietăți de bază: Title, BorderType

Un grup de obiecte, care sunt legate din punct de vedere functional, pot fi grupate vizual. Panoul are proprietatea Title, a cărei valoare sete afișată în colțul superior stânga (în variant implicită). Dacă nu se dorește să apară text, proprietății Tiltle nu i se atribuie valoare. Cu proprietatea BorderStyle se modifică aspectul. Valorile cele mai utilizate sunt beveledin sau etchedin.

4.3.10. Tabele

Descriere: Tabelul reprezintă o matrice de date, ce poate fi editată.

Clasa: uitable

Proprietăți de bază: Data, RowName, ColumnName, ColumnEditable.

Elementele conținute în tabele pot aparține unor clase diferite: valori numerice, logice sau șiruri. Existența barelor derulante pe orizontală și vertical permit vizualizarea unui volum mare de date. Se pot seta titluri pe coloane și linii, cu ajutorul matricelor de celule. Fiecare coloană poate fi declarată editabilă sau nu.

Proprietatea Data este o matrice complexă (cell array) și stochează conținutul tabelului. Tabelul poate fi încărcat cu valori și formatat în Guide, dar este mai ușor să se lucreze cu funcția CreateFcn. Această funcție asociată unui obiect din GUI, se execută la lansarea GUI. Nu se poate presupune că legătura de manevrare, pointerul către structură (handles) a fost făcut la rularea funcției CreateFcn. Pointerul către obiect, tabelul în acest caz, se transferă funcției prin argumentul hObject.

În continuare se exemplifică definirea unui tabel cu ajutorul funcției CreateFcn.

```
function studentiTable_CreateFcn(hObject, eventdata, handles)
studenti={'Ionescu','Gheorghe','9.29';'Georgescu','Mihai','7.25';…};
set(hObject,'Data',studenti);
set(hObject,'RowName',{'1','2','3','4'});
set(hObject,'ColumnName',{'Nume','Prenume','Medie'});
set(hObject,'ColumnEditable',[false,false,true])
```

Când utilizatorul modifică o valoare editabilă din tabel funcția CellEditCallback se execută. Această funcțue trebuie adusă din meniu, la fel ca în cazul grupului de butoane, prin selectarea View/View Callbacks/CellEditCallback. Când această funcție este apelată, funcția primește în plus față de pointer, o structură denumită eventdata, ce conține informații legate de celula care a fost editată, care a fost conținutul anterior și care este conținutul actual. În următorul exemplu se ilustrează cum se poate extrage această informație:

```
function studentiTable_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to studentiTable (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
%   Indices: row and column indices of the cell(s) edited
%   PreviousData: previous data for the cell(s) edited
%   EditData: string(s) entered by the user
%   NewData: EditData or its converted form set on the Data property. Empty
if Data was not changed
%   Error: error string when failed to convert EditData to appropriate
value for Data
% handles    structure with handles and user data (see GUIDATA)
iRowChanged=eventdata.Indices(1);
iColChanged=eventdata.Indices(2);
oldValue=eventdata.PreviousData;
newValue=eventdata.NewData;
disp(['Data la(',num2str(iRowChanged),',',num2str(iColChanged),') s-a
modificat de la ',num2str(oldValue),'la ',num2str(newValue)]);
```

4.3.11. Sisteme de axe

Descriere: este un obiect pentru reprezentarea grafică a informației, de tipul grafice sau imagini.

Clasa: axes

Proprietăți de bază: grid, FontSize, FontWeight, FontName, LineWidth, XTick, YTick, ZTick, Box, ButtonDownFcn, CurrentPoint.

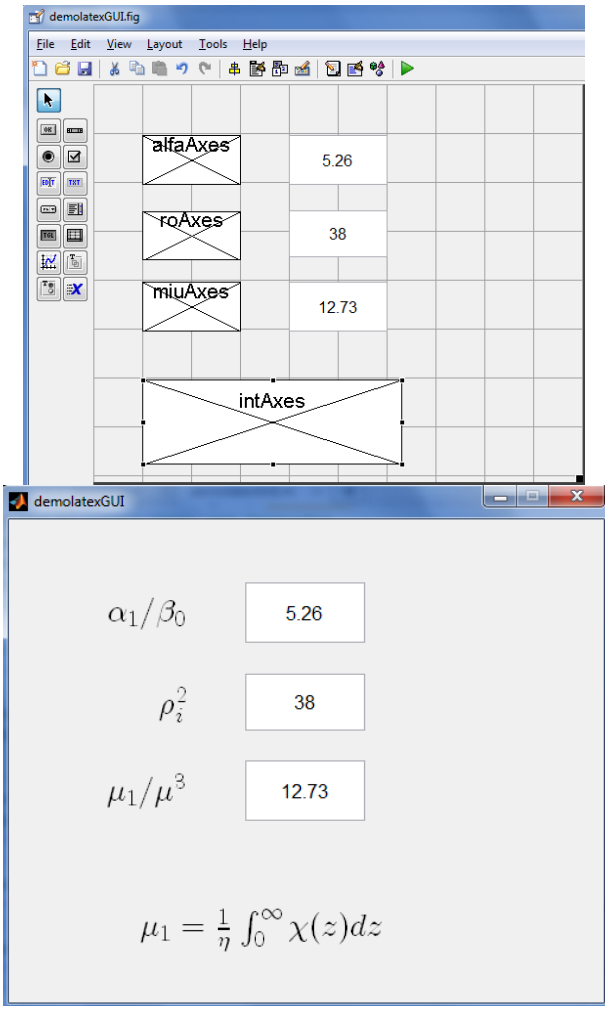
Pentru a reprezenta grafic o informație se utilizează o comandă de desenare sau una pentru imagini: plot, plot3, surf,. pcolor, image. Uneori proprietățile specifice ale sistemului de axe pot fi specificate mai ușor utilizând comenzile grafice: axis, xlabel, ylabel, zlabel, title, grid, view.

Trebuie menționat că pentru sistemul de axe, proprietatea LineWidth controlează grosimea liniilor de axă, nu a graficului propriu-zis.

Când într-o GUI există mai multe sisteme de axe este important să se trimită comenzile în mod corect.

Obiectul sistem de axe poate fi utilizat pentru etichete în loc controlului static text. Avantajul oferit este posibilitatea formatării textului afișat cu ajutorul comenzilor din Latex. Acest lucru permite tipărirea caracterelor grecești, simboluri matematice, indici și exponenți. Pentru a putea face acest lucru se scriu comenzile în funcția OpeningFcn a GUI. Pentru fiecare sistem de axe ce urmează a fi utilizat ca o etichetă trebuie parcurse etapele:

1. se suprimă axele, prin setarea proprietății axis ‚off';
2. se creează un obiect text, cu proprietatea Interpreter setată 'latex';
3. se declară obiectul text ca obiect derivat al obiectului axe (proprietatea Parent a textului este declarat sistemul de axe).



Codul aferent realizării unei afișări de forma celei din figură este:

```
axis(handles.alfaAxes,'off');
ht=text(1,0.5,'$\alpha_1/\beta_0$','HorizontalAlignment','Right','Interpreter','latex','FontSize',20);
set(ht,'Parent',handles.alfaAxes);
axis(handles.roAxes,'off');
ht=text(1,0.5,'$\rho^2_{i}$','HorizontalAlignment','Right','Interpreter','latex','FontSize',20);
set(ht,'Parent',handles.roAxes);
axis(handles.miuAxes,'off');
ht=text(1,0.5,'$\mu_1/\mu^3$','HorizontalAlignment','Right','Interpreter','latex','FontSize',20);
set(ht,'Parent',handles.miuAxes);
axis(handles.intAxes,'off');
ht=text(1,0.5,'$\mu_1=\frac{1}{\eta}\int_0^\infty\chi(z)dz$','HorizontalAlignment','Right','Interpreter','latex','FontSize',20);
set(ht,'Parent',handles.intAxes);
```

Latex este un limbaj de editare de text, utilizat pentru formatarea expresiilor matematice. Matlab are implementate o serie de comenzi de editare text (xlabel, ylabel, title, text), care interpretează automat anumite comenzi Latex. În aceste comenzii se pot obține indici sau exponenți prin utilizarea simbolurilor _ și ^. Ex. x^2 devine x^2, x^{a+b} →x^{a+b}, y_n →y_n. Textul se include între caracterele $.

Acolada se utilizează la gruparea simbolurilor. Multe din comenzile Latex încep cu caracterul \, ex. simbolul radical se declară \sqrt. Fracția se declară \frac{numerator}{numitor}. Ex.

\frac{x+1}{x-1} → $\frac{x+1}{x-1}$

\frac{d^2y}{dx^2} → $\frac{d^2y}{dx^2}$

\frac{-b\pm\sqrt{b^2-4ac}}{2a} → $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

Simbolizarea semnului de integrală se face \int, ∑ se simbolizează \sum.

\int_0^\infty f(x)dx este echivalent cu $\int_0^\infty f(x)dx$

\sum_{i=0}^n \frac{1}{x^i} $\sum_{i=0}^{n}\frac{1}{x^i}$

\lim_{x\to\infty} \frac{x}{x+a} = 1 $\lim_{x\to\infty}\frac{x}{x+a}=1$

Coduri pentru litere grecești

\alpha	α	\iota	ι	\sigma	σ	\Delta	Δ
\beta	β	\kappa	ϰ	\varsigma	ς	\Theta	Θ
\gamma	γ	\lambda	λ	\tau	τ	\Lambda	Λ
\delta	δ	\mu	μ	\upsilon	υ	\Xi	Ξ
\epsilon	ϵ	\nu	ν	\phi	ϕ	\Pi	Π
\varepsilon	ε	\xi	ξ	\varphi	φ	\Sigma	Σ
\zeta	ζ	\pi	π	\chi	χ	\Upsilon	Υ
\eta	η	\varpi	ϖ	\psi	ψ	\Phi	Φ
\theta	θ	\rho	ρ	\omega	ω	\Psi	Ψ
\vartheta	ϑ	\varrho	ϱ	\Gamma	Γ	\Omega	Ω

4.4. Utilizarea meniurilor și a casetelor de dialog în GUI

În MATLAB există o serie de instrumente GUI predefinite:

- bara de urmărire (waitbar) – ce permite urmărirea progresului unei operații;
- casete de dialog de tip intrare sau input și de tip întrebare sau chestionar;

- se poate dialoga cu fișiere prin schimbarea de informații (se trimite informație în fișier sau se extrage informație);
- se pot construi aplicații, care să răspundă la evenimente declanșate de mouse sau tastatură;
- se pot construi meniuri.

4.4.1. Bara de așteptare sau de urmărire (waitbar)

Bara de urmărire se utilizează pentru a indica evoluția în timp a unei sarcini, Cea mai uzuală aplicație este urmărirea evoluției unei bucle for, care este consumatoare de timp. În primul rând trebuie apelată comanda waitbar, înainte de începerea buclei, pentru a afișa valoarea inițială 0 și a stoca pointerul barei. Sintaxa este:

<handle>=waitbar(0,'sir mesaj','Name','titlu');

unde <handle> va fi pointerul către bară, sir mesaj este textul ce se va afișa ăn casetă, iar titlu va apare în partea superioară a casetei.

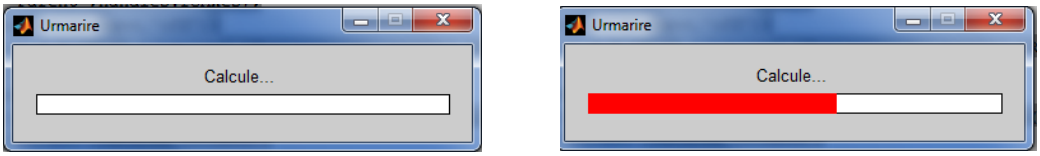
La fiecare iterație comanda este apelată pentru a actualize evoluția:

waitbar(progres, <handle>);

iar la final, bara se șterge cu comanda:

close(<handle>);

Ex.




```
hw=waitbar(0,'Calcule...','Name','Urmarire');
for k=1:N
      % instructiuni de calcul
      waitbar(k/N,hw)
end
close(hw)
```

4.4.2. Dialogul cu fișiere

O primă problemă abordată este salvarea și încărcarea informației în fișiere cu extensia .mat. Comanda save salvează valorile și numele variabilelor MATLAB. Sintaxa este:

```
save Nume_fisier lista_variabile
```

sau

```
save('Nume_fisier','var1','var2','var3');
```

De ex. variabilele x, y și t, ce pot fi valori scalare sau variabile de tip matricial, se salvează într-un fișier myData

```
save MyData x y t
save('MyData','x','y','t');
```

A doua variantă a comenzii este necesară dacă numele fișierului este stocat într-o variabilă de tip șir.

Ex.

```
f_nume='myData';
save(f_nume,'y','y_init','yfinal');
```

Fișierul myData.mat este un fișier specific MATLAB, este binar, nu este fișier text și, în general, nu poate fi citit decât de MATLAB. Fișierul conține numele și clasa fiecărei variabile, precum și valorile asociate. Dacă nu se specifică o listă de variabile, toate variabilele din spațiul de lucru se vor salva.

Citirea informației din fișiere de tip .mat se poate face cu comanda load:

```
load myData  %incarca toate variabilele din myData.mat
load myData x y  % incarca variabilele x si y din myData.mat
```

sau

```
fisier='myData';
load(fisier);
load(fisier,'x','y');
```

Numele fișierului poate include calea spre fișier:'D:\Documents\Exemple\Curs\myData', ceea ce permite ca fișierul să se găsească și în alt director decât directorul curent.

În cazul când o variabilă cu același nume exista în momentul încărcării fișierului, valoarea variabilei se va prelua din fișier.

În Windows utilizatorii sunt obișnuiți să navigheze prin directoare pentru a găsi un fișier, să atribuie nume fișierelor. O astfel de posibilitate este oferită de comanda uiputfile – pentru salvare și uigetfile – pentru încărcare. Este important să se înțeleagă că aceste comenzi oferă posibilitatea de obținere a numelui fișierului și nu salvează sau încarcă informația. Sintaxa este:

```
[NumeFisier,Cale,index_filtru]=uiputfile(filtru,titlu,Nume_implicit)
```

unde:

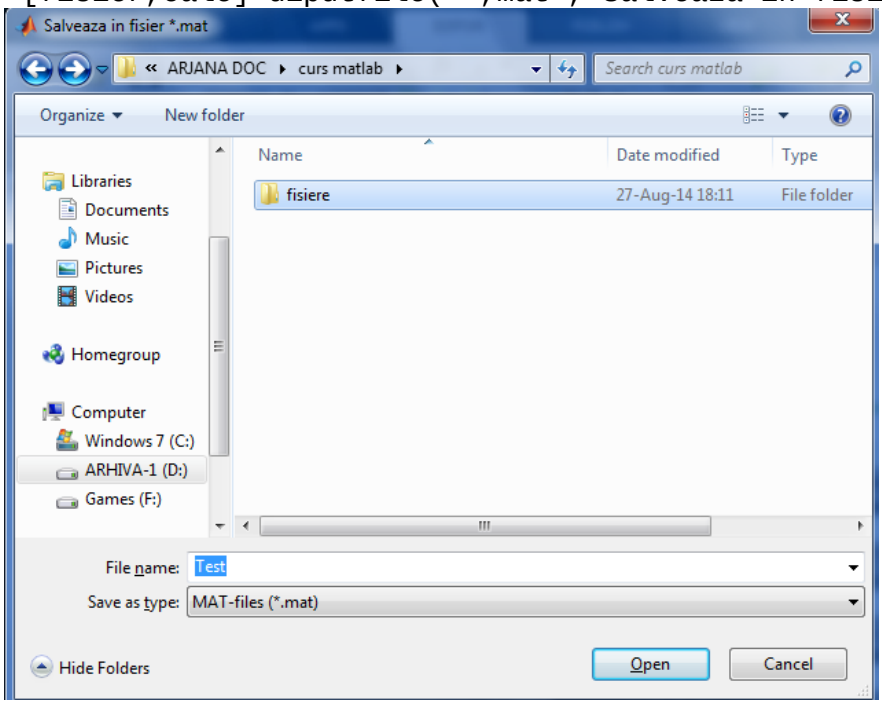
- filtru determină fișierele afișate inițial în fereastra de dialog, '*.m' va afișa toate tipurile de fișiere MATLAB.
- titlu este titlul ferestrei

Nume_implicit este numele fișierului afișat la deschiderea ferestrei.

Funcția returnează numele fișierului așa cum este specificat de utilizator, calea spre fișierul respectiv și indexul filtrului care începe de la 1.

Ex.

```
[fisier,cale]=uiputfile('*,mat','Salveaza in fisier .mat','Test');
```



Funcția determină apariția unei ferestre de dialog, ca în figură, în care se poate naviga, se poate creea un director nou și atribui orice nume valid fișierului. Pentru a face salvarea propriu-zisă trebuie să urmeze comanda:

```
save([cale,fisier],'x','y');
```

Primul argument al comenzii este numele complet cu calea și se formează prin concatenare. Atenție: ordinea celor două variabile este inversă față de comanda uiputfile.

Pentru a încărca un fișier se utilizează comanda uigetfile, care lucrează similar. Ea are sintaxa:

```
[Nume_fisier,Cale,Index_filtru]=uigetfile(filtru,titlu,nume_implicit
);
```
Semnificația variabilelor este aceeași ca la funcția uiputfile. La fel ca la salvare, pentru a încărca informația din fișier trebuie apelată comanda load:

```
load([Cale,Nume_fisier]);
```

4.4.3. Citirea și scrierea în fișiere formatate text

Fişierele de tip .mat stochează variabile le MATLAB şi valoarea acestora. O altă variantă este scrierea informaţiei şi citirea din fişiere text. Avantajul fişierelor text este că pot fi vizualizate şi de late programe. Dezavantajul este că nu se reţine numele variabilei şi clasa acesteia. De asemenea, în funcţie de modul de scriere al textului se poate pierde din precizie.

Pentru a scrie date în fişier de tip text trebuie parcurşi trei paşi:

1. se deschide fişierul cu comanda fopen, aceasta asociază o valoare întreagă, numărul de identificare al fişierului, acestuia. Fişierul se poate deschide în mai multe moduri de acces.
2. Se scrie informaţia în fişier cu comanda fprintf. Informaţia va fi scrisă sub formă de şir formatat.
3. Se închide fişierul cu comanda fclose.

Comanda fopen deschide fişierul într-un anume mod de acces şi returnează numărul de identificare al fişierului. Sintaxa este:

```
fid=fopen(,NumeFisier.txt',’<SirAcces>’);
```

De ex.: următorul cod defineşte o variabilă de tip întreg, a, două variabile reale, b şi c, şi o variabilă de tip şir; deschide un fişier şi adaugă o linie în fişier cu valoarea celor patru variabile.

```
a=randi([1,10]);
b=7.2*sqrt(2);
c=3*pi;
nume=’Adriana’;
fid1=fopen(‘fisiertest.txt’,’a’);
fprintf(fid1,’%d %f %5.3f %s\n’,a,b,c,nume);
fclose(fid1)
```

Rezultatul va fi următoarea linie adăugată la finalul fişierului fisiertest.txt (dacă acesta nu există, va fi creat):

8 10.182338 9.425 Adriana

Fişierul a fost deschis în modul „append” (adăugare). Modurile de acces pot fi:

- scriere ‚w’ – se scriu date noi, conţinutul anterior se pierde;
- adăugare ‚a’ – se adaugă la sfârşitul fişierului, deci se menţine conţinutul anterior;
- Citire ‚r’ – nu se stochează informaţie, doar se citeşte.

Sintaxa pentru comanda fprintf (file print formatted) este:

```
fprintf(fid,’<sir de formatare>’,var1,var2,...);
```

unde fid este numărul de identificare al fişierului, returnat de fopen. Sir de formatare controlează modul în care valorile variabilelor vor apare în fişier. Dacă în sintaxă lipseşte fid, tipărirea se va face în fereastra de comandă.

Ex.

```
x=1.2;
fprintf(’Inaltime=%g metri\n’ x)    →Inaltime=1.2 metri
```

Atât %f, cât şi %g sunt formate pentru numere reale, diferenţa este că %g nu va tipări decât până la ultima cifră semnificativă (taie zerourile de la sfârşit).

%d se utilizează pentru valori întregi, iar s pentru şiruri

```
fprintf(’%d %18.15f %f \n Nume: %s\n’,a,b,c,nume)
→ 8 6.521701946944458 9.424778 Nume: Adriana
```

Prima valoare numerică, 18, din faţa caracterului f reprezintă numărul de digiţi pentru reprezentare (inclusiv punctul zecimal şi spaţiile dinainte), din care a doua valoare, 15, este numărul de digiţi pentru partea fracţionară; ’%8.5g’ indică 5 digiţi pentru partea fracţionară din câmpul total de 8 digiţi.

Dacă în lista de variabile apare o matrice, ea va fi tipărită pe coloane, dar nu sub formă de matrice, ci ca un şir de elemente, coloană după coloană, de sus în jos. Pentru a putea tipări matricea corect trebuie utilizate atâtea simboluri %d câte coloane are matricea, dar datorită ordinii de tipărire, ea va apare ca matricea transpusă.

Fie matricea:

```
a=[1,2,3;4,5,6;7,8,9];
```
$\rightarrow a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

```
>> fprintf('%d \n',a)
1
4
7
2
5
8
3
6
9
>> fprintf('%d %d %d\n',a)
1 4 7
2 5 8
3 6 9
```

Pentru a vedea corect matricea trebuie cerută tipărirea matricei transpuse:

```
>> fprintf('%d %d %d\n',a')
1 2 3
4 5 6
7 8 9
```

Citirea informației dintr-un fișier text poate fi complicată și modul cum a fost scrisă informația trebuie cunoscut dinainte. Pentru a citi datele dintr-un fișier, acesta poate fi deschis în modul de acces ‘r’ (read), iar citirea se poate face cu comanda textscan. Sintaxa comenzii este:

variabile=textscan(fid,’sir formatare’);

unde fid este numărul de identificare a fișierului, iar șirul de formatare are aceeași semnificație ca în cazul comenzii fprintf.

În Matlab se pot importa şi exporta fişiere de tipul ”comma separated values” sau formatul csv. Acestea sunt fişiere ASCII, ale căror valori de pe fiecare linie sunt separate cu virgulă, după cum sugerează şi titlul. Sintaxa este:

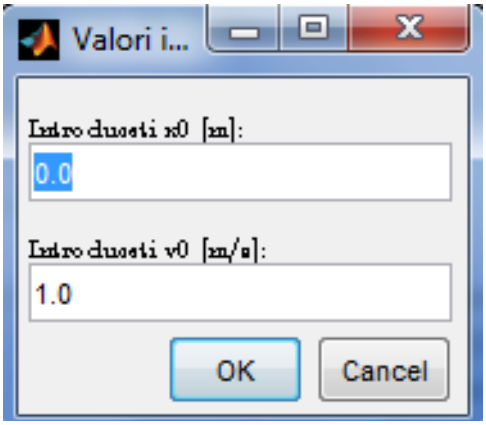
A=csvread(‘nume_fisier’) – citeşte fişierul şi introduce valorile în A. Fişierul trebuie să conţină numai valori numerice.

csvwrite(,nume_fisier,A) – scrie matricea A într-un fişier cu extensia csv.

4.4.4.Caseta de dialog Input și Question

Caseta de dialog predefinită, inputdlg, permite introducerea de către utilizator a uneia sau mai multor valori, returnând răspunsul utilizatorului într-o matrice complexă, de tip cell array. De ex. următorul cod oferă posibilitatea introducerii a două valori:

```
comentariu={'Introduceti x0 [m]:','Introduceti v0 [m/s]:'};
nume='Valori initiale';
numarlinii=1;
implicit={'0.0',1.0'};
options.Resize='on';
options.Interpreter='latex';
caStr=inputdlg(comentariu,nume,numarlinii,implicit,options)
```



Caseta de dialog Input

Se observă semnificația variabilelor:

- comentariu este textul explicativ ce apare în casetă, este o matrice complexă ce conține un număr de elemente egal cu numărul de valori pe care utilizatorul trebuie să le introducă:
- nume este textul ce apare în bara de titlu a casetei;
- numar linii reprezintă numărul de linii alocat zonei în care utilizatorul va introduce valori; este valoare întreagă;
- implicit, este o matrice complexă, fiecare element fiind de tip șir. Reprezintă textul ce apare inițial ca sugestie pentru utilizator.
- options este o structură de date ce permite redimensionarea casetei și utilizarea interpretorului latex.

Funcția returnează o matrice complexă, caStr, ce conține valorile introduse de utilizator, în cazul când acesta va tasta OK. Dacă se tastează Cancel, se va returna un șir de lungime nul (de lungime 0).

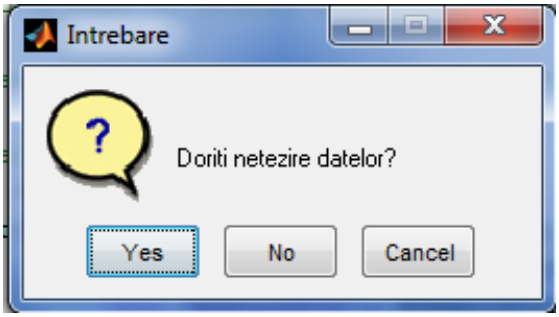
```
if length(caStr)>0
      x0=str2double(caStr{1});
      v0= str2double (caStr{2});
      disp(['x0= ',caStr{1},'v0= ',caStr{2})
end
```

Caseta de dialog pentru întrebări (question dialog) permite obținerea unui răspuns binar (da/nu) de la utilizator, iar comanda are sintaxa:

questdlg('Intrebare','Titlu');

unde Intrebare este întrebarea adresată, iat Titlu este titlul casetei. Funcția returnează un șir: 'Yes', 'No' sau 'Cancel'.

```
raspuns=questdlg('Doriti netezire datelor?','Intrebare');
optiune=false;
switch raspuns
      case 'Yes'
            optiune='true';
      case 'No'
            disp('Nu se filtreaza datele');
      case 'Cancel'
            disp('Nu se filtreaza datele');
end
```



4.4.5. Partajarea datelor între funcţii

Intr-o interfaţă grafică utilizator se pot seta anumite valori ale unei proprietăţi cu comanda set şi se pot prelua cu comanda get. Pot apare situaţii când rezultatul unui volum mare de calcule să trebuiască să fie accesat de alte funcţii. Comenzile setappdata şi getappdata lucrează similar cu set şi get, dar pot opera asupra unor proprietăţi definite de utilizator. Datele pot fi asociate cu orice obiect.

De ex. Într-o GUI există un sistem de axe în care se reprezintă traiectoria unui obiect pe baza unor şiruri x, y şi t. Cu funcţia gcf (get current figure) se obţine handler-ul pentru figură, iar datele se consideră aplicate pe figură.

```
traj.x=xBall;
traj.y=yBall;
traj.t=t;
setappdata(gfc,'Traiectorie',traj);
```

4.4.6. Răspunsul la acţionarea tastaturii

In multe situaţii este util ca programul să răspundă la acţionarea anumitor taste, nu numai la mouse. Fereastra grafică are proprietatea KeyPressFcn. Dacă valoarea acestei proprietăţi este setată la handler-ul unei funcţii, funcţia va fi executată ori de câte ori tastatura este acţionată. Funcţia de prelucrare a apăsării tastei trebuie să aibe minim două argumente. Primul argument este handler-ul figurii propriu-zise. Al doilea argument este o structură eveniment, ce are mai multe câmpuri: pentru caracterul tastat, tasta acţionată sau modificatorul acţionat (Shift, Alt, Ctrl), ce sunt reprezentate în cod prin şiruri într-o matrice complexă, de ex. {'shift', 'control'}. Săgeţile şi tastele din zona numerică au coduri distincte.

Pentru ca funcţia de prelucrare a tastaturii asociată cu fereastra să fie apelată, trebuie ca fereastra să fie activă. Acest lucru se realizează prin selectarea ferestrei.

Următoarea secvenţă de cod utilizează o fereastră de reprezentare grafică goală şi demonstrează extragerea informaţiei legată de tasta acţionată din structura eveniment evnt, care este transferată funcţiei de prelucrare a tastelor acţionate. Funcţia castr concatenează şirurile din matricea complexă evnt.Modifier într-un singur şir, pentru a-l afişa.

```
function DemoKeyPress
figure(1)
set(gcf,'KeyPressFcn',@processKey);
%% activeaza fereastra prin executare click
function processKey(src,evnt)
if length(evnt.Character)>=1
    clc
    disp(['Caracter: ',evnt.Character]);
    disp(['Modificator: ',ca2str(evnt.Modifier)]);
    disp(['Tasta: ',evnt.Key]);
end
function cas=ca2str(ca)
% converteste 1xN matrice complexa continand siruri intr-un singur sir
[n1,n]=size(ca);
cas='';
```

```
for k=1:n
    cas=[cas,' ',ca{k}];
end
```

De ex., dacă se rulează aplicaţia, se selectează fereastra grafică şi se tastează shift+t, în fereastra de comandă va apare textul:

```
Caracter: T
Modificator:  shift
Tasta: t
```

4.4.7. Interacţiunea cu obiectele grafice

Există o serie de funcţii asociate cu obiectele grafice, funcţii care se pot activa. Prin setarea corespunzătoare a acestor funcţii obiectele grafice pot deveni interactive, interceptând evenimente ca executarea unui click pe obiect sau tragerea obiectului cu mouse-ul (drag & drop).

Următoarea aplicaţie demonstrează cum trebuie setată proprietatea ButtonDownFcn pentru a creea o funcţie Callback care să fie apelată când se execută click cu mouse-ul deasupra obiectului. Pentru simplitate se construieşte un obiect triunghiular într-o figură, fără a utiliza o GUI. La executarea unui click deasupra obiectului culoarea triunghiului se modifică din roşu în albastru, iar poziţia centrului de greutate este afişat în fereastra de comandă.

```
function DemoButtonDown
% construieste un petec triunghiular ce isi modifica culoarea la click
%% construieste axe
hf=figure(1);
hax=axes;
axis(hax,[0,10,0,10]);
%% definire varfuri triunghi
x=[4,6,5,4];
y=[3,3,5,3];
%% desenare triunghi si setare functie ButtonDownFcn
hp=patch(x,y,'r');
set(hp,'ButtonDownFcn',@patch_ButtonDownFcn);
function patch_ButtonDownFcn(hObject,eventData);
x=get(hObject,'XData');
y=get(hObject,'YData');
xmid=sum(x)/length(x);
ymid=sum(y)/length(y);
disp(['Obiect centrat in x= ',num2str(xmid),' ','y= ',num2str(ymid)]);
%% nodificare culoare triunghi
culoare=get(hObject,'FaceColor');
if culoare==[1,0,0] % vector culoare este [Red,Green,Blue]
    culoare=[0,0,1];
elseif culoare==[0,0,1]
    culoare=[1,0,0];
end
set(hObject,'FaceColor',culoare);
```

Prin utilizarea unei combinaţii de funcţii callback asociate acţiunilor mouse-ului, un obiect grafic poate fi tras cu ajutorul mouse-ului. Pentru a realiza acest lucru trebuie parcurse trei etape:

1. Se execută click stânga cu mouse-ul când cursorul este deasupra obiectului.
2. Cu butonul apăsat, se deplasează mouse-ul, obiectul este mutat astfel încât cursorul rămâne în acelaşi punct din obiect. Această mişcare lasă impresia că obiectul este lipit de mouse.
3. La eliberarea butonului obiectul nu se mai deplasează.

Primul eveniment este interceptat prin setarea proprietăţii ButtonDowmFcn la valoarea pointerului către funcția asociată cu obiectul grafic. Evenimentul doi şi trei sunt asociate cu fereastra grafică.

O fereastră grafică are trei proprietăţi asociate cu evenimente ale mouse-ului :

- WindowButtonDownFcn – se apelează la apăsarea butonului de la mouse;
- WindowButtonMotionFcn – se apelează când cursorul se mută în fereastră;
- WindowButtonUpFcn – se apelează la eliberarea butonului de la mouse.

In mod implicit, fiecare proprietate are o valoare nulă. Fiecare tip de eveniment poate fi asociat cu o funcţie de procesare a evenimentului prin setarea proprietăţii ca fiind pointerul către funcţie. Fiecare funcţie are minim două argumente: pointerul către obiect (figura sau obiectul grafic din figură) şi structura evenimentului (aici nu se utilizează). Argumentele suplimentare pot fi trimise spre funcţie printr-o matrice complexă ce va conţine pointerul şi alte valori.
Următoarea aplicaţie poate fi utilizată ca un şablon pentru a face obiectele deplasabile cu mouse-ul sau cu săgeţile de la tastatură.

```
function dragDemo2
%% demonstreaza tragerea obiectelor grafice cu mouse si taste
figure(1)
%% desenare obiect
axis([0,10,0,10]);
hpatch(1)=patch([4,4,6,6,4],[1,3,3,1,1],'r');
hpatch(2)=patch([7,7.5,8,7],[1,2,1,1],'b')
%% setare callback pentru button down,up si keypress
set(hpatch(1),'ButtonDownFcn',@patchButtonDown);
set(hpatch(2),'ButtonDownFcn',@patchButtonDown);
 set(gcf,'WindowButtonUpFcn',@ButtonUp);
set(gcf,'KeyPressFcn',@procesKey);
%% Functii buton
function patchButtonDown(hpatch,eventdata)
%la executare click se seteaza functia de deplasare
%determinare pozitie petec
x=get(hpatch,'XData');
y=get(hpatch,'YData');
% pozitionare fata de cursor - referinta este primul varf
ref=[x(1),y(1)];
punct=get(gca,'CurrentPoint'); %coordonate poz cursor
distanta=[punct(1,1),punct(1,2)]-ref;
%% specificare functie WindowButtonMotionFcn cu 2 argumente suplimentare
handle patch si distanta
setappdata(gcf,'SelectedPatch',hpatch);
set(gcf,'WindowButtonMotionFcn',{@figButtonMotion,hpatch,distanta});
function ButtonUp(src,eventdata);
% la eliberare buton se se opreste deplasarea
set(gcf,'WindowButtonMotionFcn',' ');
function figButtonMotion(src,eventdata,hselected,distanta)
% deplaseaza obiectul cu handle hselected
% determinare pozitii varfuri
x=get(hselected,'XData');
y=get(hselected,'YData');
% coordonate fata de primul varf
xr=x-x(1);
yr=y-y(1);
% coordonate pozitie cursor
punct=get(gca,'CurrentPoint');
%modificare pozitie
set(hselected,'XData',punct(1,1)-distanta(1)+xr);
set(hselected,'YData',punct(1,2)-distanta(2)+yr);
function procesKey(src,evnt)
% deplaseaza obiect cu sageti tastatura
hpatch=getappdata(gcf,'SelectedPatch');
if length(evnt.Key)>=1
    xshift=0;
    yshift=0;
    switch evnt.Key
```

```
        case 'leftarrow'
            xshift=-0.1;
        case 'rightarrow'
            xshift=0.1;
        case 'uparrow'
            yshift=0.1;
        case 'downarrow'
            yshift=-0.1;
    end
    x=get(hpatch,'XData')+xshift;
    y=get(hpatch,'YData')+yshift;
    set(hpatch,'XData',x);
    set(hpatch,'YData',y);
end
```

Funcția principală dragDemo2 desenează un petec pătrat și unul triunghi, proprietatea ButtonDownFcn a petecului este setată pointerul către funcția patchButtonDown, ceea care se activează la executarea click pe petec. Această funcție, patchButtonDown, are ca argument pointerul către petecul selectat și o structură eveniment. Proprietatea WindowButtonUpFcn a ferestrei grafice este pointerul către funcția ButtonUp, iar proprietatea KeyPressFcn a ferestrei este pointerul către funcția processKey.

La invocarea funcției patchButtonDown, aceasta primește poziția cursorului prin intermediul proprietății CurrentPoint a sistemului de axe (pointerul către sistemul de axe se generează prin funcția gca – get current axes). Se calculează vectorul distanță, ca diferență între poziția cursorului și primul vârf al petecului. Acest vârf servește ca referință pentru petec.

Există posibilitatea ca mai multe obiecte grafice să fie combinate astfel încât să poată fi tratate ca un singur obiect, ce poate fi translatat sau rotit în spațiu, poate fi scalat etc.. Obiectul compus se numește obiect hgtransform. Acesta poate conține mai multe obiecte derivate, orice obiect ce poate fi derivat al unui sistem de axe, cu excepția obiectelor luminoase. De asemenea, imaginile nu pot fi transformate, ele nefiind obiecte reale 3D. Creearea obiectului compus presupune parcurgerea unor pași:

- se definește fiecare obiect individual (suprafață, text, linie etc.), iar pointerii pentru aceste obiecte se consideră hobiect1, hobiect2, etc.;
- se construiește obiectul hgtransform, inițial fără să conțină ceva;
- opțional, se poate specifica sistemul de axe de care aparțin toate obiectele (trebuie să aparțină unui sistem de axe unic, nu pot fi derivate din mai multe sisteme);
- se declară obiectele individuale ca fiind derivate ale obiectului hgtransform, prin setarea proprietății Parent la pointerul către obiectul compus.

Ex.

```
hl=line(x,y,'Color','r');
hp=patch(x1,y1,'r');
hcompus=hgtransform('Parent',gca);
set(hl,'Parent',hcompus);
set(hp, 'Parent',hcompus);
```

Proprietatea Children a obiectului hgtransform va fi automat setată la un vector coloană ce conține toți pointerii către fiecare obiect individual. Manipularea obiectului hgtransform se face prin intermediul proprietății Matrix, o matrice (4, 4). În MATLAB există implementată funcția makehgtform care returnează matricea corespunzătoare unei rotații în jurul unei axe, o translație de-a lungul unui vector sau scalării unui obiect. Matricele corespunzătoare diferitelor transformări și sintaxa corespunzătoare funcției makehgtform sunt prezentate în continuare:

Rotire în jurul axei 0x cu unghiul α

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M=makehgtform('xrotate',alfa);

Rotire în jurul axei 0y cu unghiul α

$$M = \begin{bmatrix} cos\alpha & 0 & sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -sin\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ M=makehgtform('yrotate',alfa);

Rotire în jurul axei 0z cu unghiul α

$$M = \begin{bmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ M=makehgtform('zrotate',alfa);

Se poate defini și rotirea în jurul axei (ax,ay,az) cu unghiul t exprimat în radiani
M=mahehgtform('axisrotate',[ax,ay,az],t);
Translație cu dx,dy,dz

$$M = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ M=makehgtform('translate',[dx.dy.dz]);

Scalare cu factori (sx, sy, sz)

$$M = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ M=makehgtform('scale',[sx,sy,sz]);

Transformările pot fi combinate prin înmulțirea matricelor de la dreapta spre stânga (de la ultima transformare spre prima). Fie următoarea secvență de mișcare:

a. translație cu -1 pe axa 0z;
b. rotire cu π/2 în jurul axei 0x;
c. translație cu +1 pe axa 0z.

Matricele corespunzătoare se obțin:

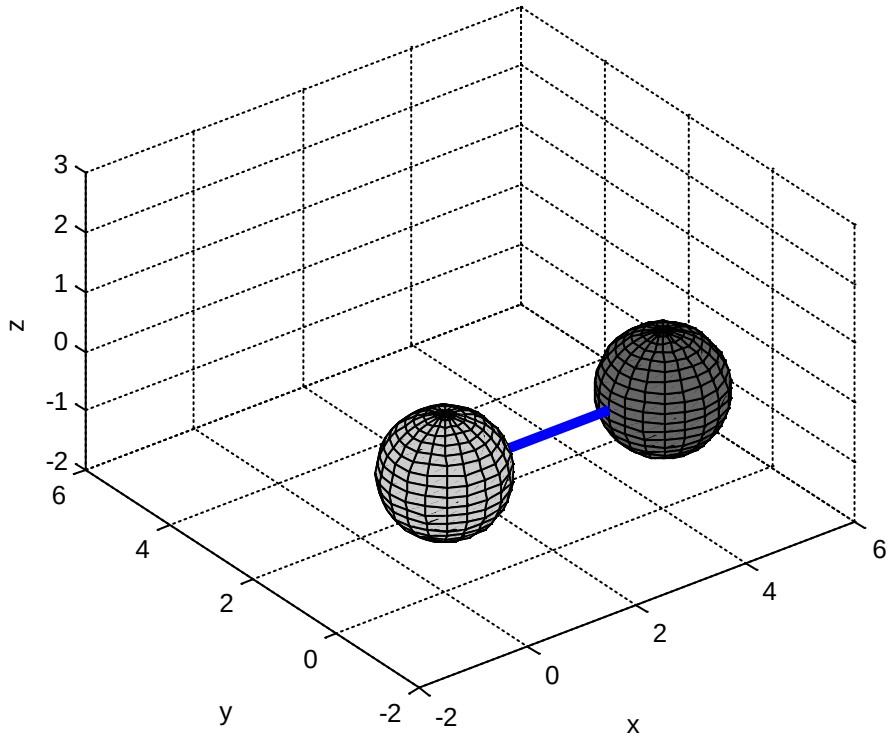
M1=makehgtform('translate',[0,0,-1]);
M2=makehgtform('xrotate',pi/2);
M3=makehgtform('translate',[0,0,1]);
set(hcompus,'Matrix',M3*M2*M1)

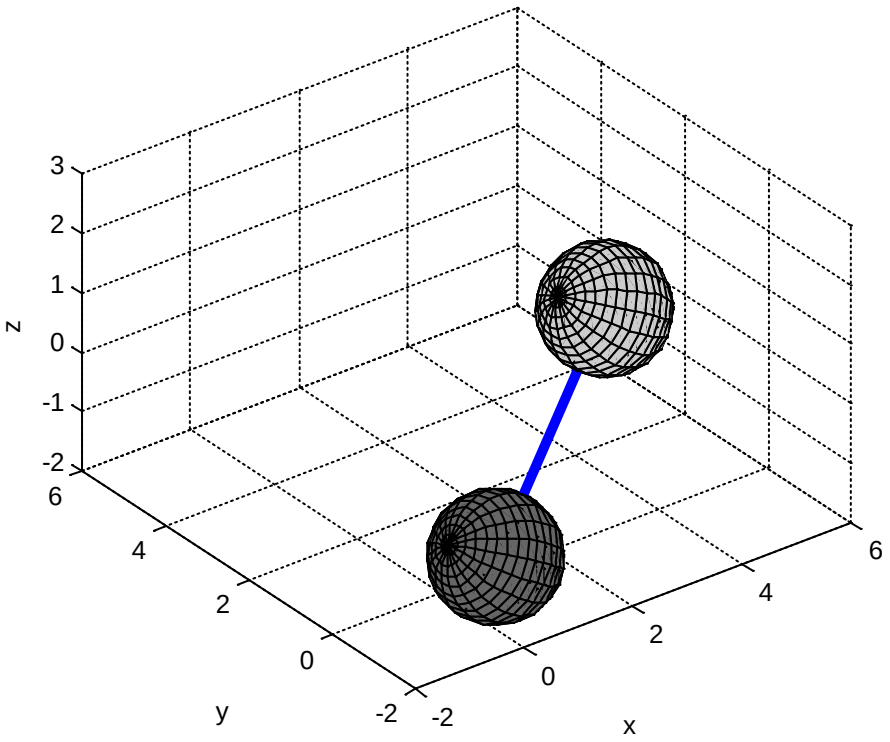
Realizarea unor transformări suplimentare se poate face extrăgând valoarea curentă a proprietății Matrix a obiectului, multiplicarea acesteia cu matricea de transformare și setarea proprietății Matrix la noua valoare. Prin setarea proprietății Matrix la matricea unitară (eye(4)) se anulează transformările anterioare.

În următoarea aplicație se formează un obiect compus din două sfere unite cu o linie, care se rotesc în jurul centrului de greutate. Deoarece centrul de rotație diferă de originea sistemului de axe, se va face o translație până în origine, se face rotirea și obiectul este translatat înapoi.

```
%% rotire_haltera.m
%se obtine un obiect compus din 2 sfere si o linie care se roteste in
%jurul centrului propriu
%% desenare obiecte
[X,Y,Z]=sphere(20);
h1=surf(X,Y,Z,'FaceColor',[0.8,0.8,0.8]);%prima sfera are centrul in
origine, raza =1
axis equal
axis([-2,6, -2,6,-2,3]);
hold on
```

```
h2=surf(X+4,Y,Z,'FaceColor',[0.4,0.4,0.4]);% a doua sfera are centrul in
(4,0,0)
hold on
hl=line([1,3],[0,0],[0,0],'LineWidth',4);
xlabel('x'),ylabel('y'),zlabel('z')
%% compunere obiect
hhaltera=hgtransform('Parent',gca);
set(h1,'Parent',hhaltera);
set(h2,'Parent',hhaltera);
set(hl,'Parent',hhaltera);
%% rotire in jurul (2,0,0)
cg=[2,0,0];
n=60;
teta=linspace(0,2*pi,n);
for i=1:n
    M1=makehgtform('translate',-cg);
    M2=makehgtform('yrotate',teta(i));
    M3=makehgtform('translate',cg);
    set(hhaltera,'Matrix',M3*M2*M1);
    drawnow
end
```





Pentru ca astfel de reprezentări să poată fi incluse în GUI, codul trebuie să fie inclus într-o funcție ce va fi apelată de funcția callback a GUI. Structura de pointeri va fi disponibilă, astfel încât pointerul către fiecare sistem de axe poate fi obținut.

În cazul comenzilor de desenare de nivel superior, pointerul către sistemul de axe apare ca prim argument al comenzii. Ex.:

```
plot(handles.AxesE,t,E1,t,E2):
xlabel(handles.AxesE,'Timp [s]');
ylabel(handles.AxesE,'Energie [J]');
grid(handles.Axes1,'on');
surf(handles.Axes2,X,Y,Z);
```

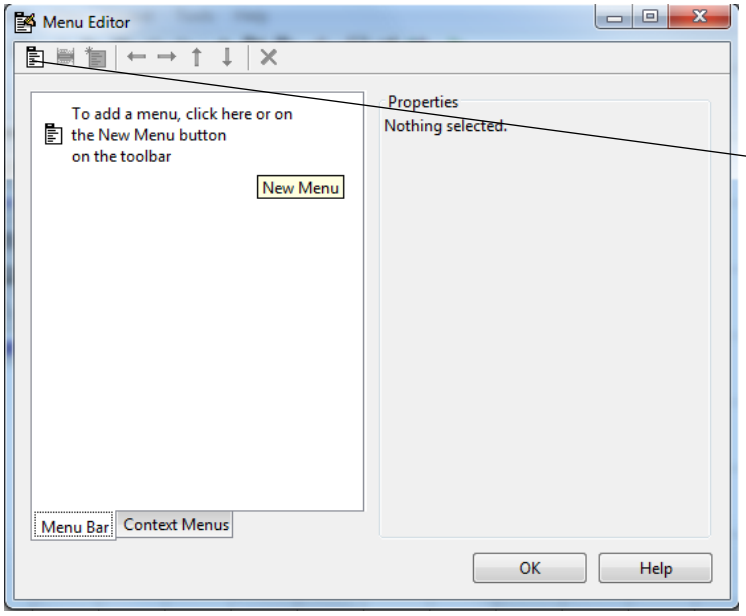
Pentru comenzi de nivel inferior, sistemul de axe trebuie definit ca 'Parent' a obiectului grafic. Ex.:

```
hlinie=line(x,y);
set(hlinie,'Parent',handles.Plot1,'Color','b');
htext=text(x1,y1,'Punct de inflexiune');
set(htext,'Parent',handles.Axes1);
him=image('NumeFisier.jpg');
set(him,'Parent',handles.ImAxes);
```

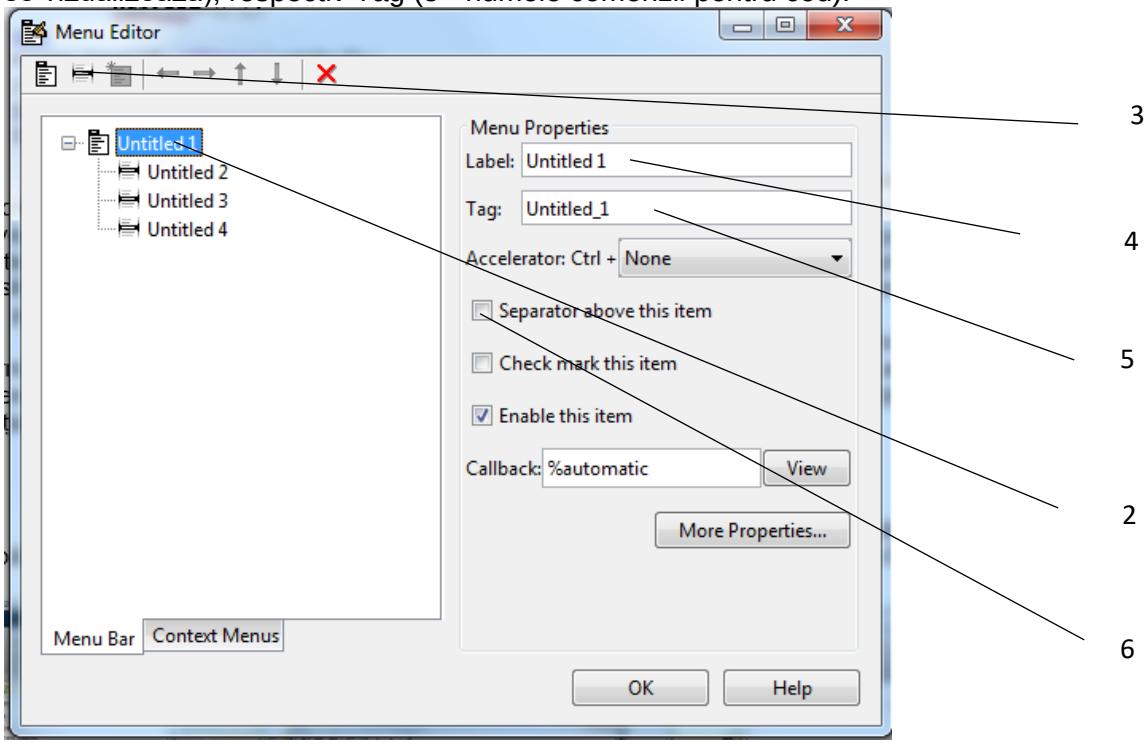
4.4.8. Creearea meniurilor în GUIDE

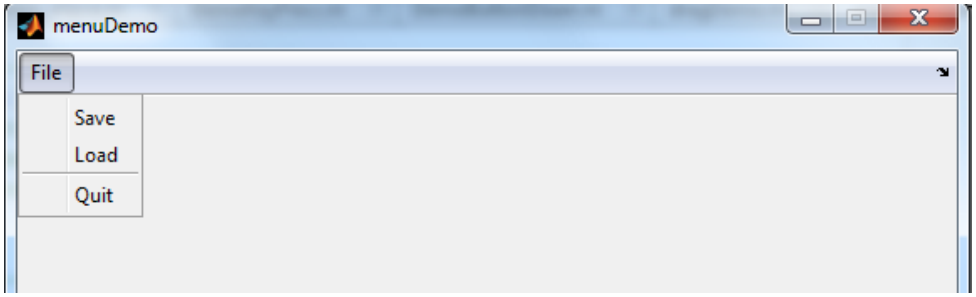
Bara de meniu este o caracteristică standard în Windows şi Guide oferă o modalitate simplă de a introduce meniuri în GUI. Instrumentul utilizat este editorul de meniuri – Menu Editor.

Se lansează GUIDE şi din meniul Tools se selectează Menu Editor.



Se va deschide editorul din figură. Pentru a defini un nou meniu se selectează prima pictogramă (1). Va apare meniul Untitled1. Dacă se doreşte ca acest meniu să conţină trei comenzi se va executa de trei ori click pe pe a doua pictogramă (3), având meniul selectat (2). In partea dreaptă a editorului apar casete pentru a defini proprietăţiile Label (4 - ceea ce se vizualizează), respectiv Tag (5 - numele comenzii pentru cod).





Se completează proprietăţiile: Save (Label) şi Save Menu (Tag), respectiv Load, LoadMenu şi Quit, QuitMenu. Prin bifarea casetei de validare Separator above this item (6) va apare o linie de separaţie în meniu. La salvarea interfeţei, în cod apar, generate automat, patru funcţii callback.

```
function Untitled_1_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function SaveMenu_Callback(hObject, eventdata, handles)
% hObject    handle to SaveMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function LoadMenu_Callback(hObject, eventdata, handles)
% hObject    handle to LoadMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function QuitMenu_Callback(hObject, eventdata, handles)
% hObject    handle to QuitMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Pentru a închide GUI se introduce în funcţia callback a meniului quit comanda:

```
close(gcf)
```