3. Programarea în Matlab

3.1. Instrucţiuni de control

Dezvoltarea aplicaţiilor în programare necesită o serie de comenzii care să permită controlul succesiunii instrucţiunilor. Instrucţiunile de control oferă flexibilitate aplicaţiilor, permiţând ramificarea, modificarea ordinii de executare a anumitor comenzi, repetarea anumitor secvenţe.

Instrucţiunile de control sunt de două tipuri:

- de decizie;
- repetitive.

INSTRUCŢIUNI DE DECIZIE

Structurile de decizie permit executarea anumitor secvenţe de cod pe baza unor condiţii. Aceste condiţii trebuie specificate în faza de concepţie şi sunt evaluate în timpul rulării. Majoritatea acestor structuri sunt de forma If…End. În cazul când condiţia specificată este adevărată, se execută blocul de instrucţiuni din structură.

Cea mai simplă variantă de testare a unei condiţii este varianta instrucţiunii if având următoarea sintaxă:

```
if <condiţie logică>
    bloc de instrucţiuni
end
```

O altă posibilitate este:

```
if <condiţie logică>
    bloc de instrucţiuni 1
else
    bloc de instrucţiuni 2
end
```

Blocul 1 de instrucţiuni se execută în situaţia când condiţia este adevărată, iar blocul 2, dacă aceasta este falsă.

Ex. Aplicaţie pentru a verifica dacă un număr introdus de utilizator este prim sau nu.

```
%% nr_prim.m
% verifica daca un numar este prim sau nu
%% date
disp('Verificare mumere prime')
n=input('Introduceti un numar: ');
if isprime(n)
    disp(['Da! ',num2str(n),' este numar prim!'])
else
    disp(['Nu! ',num2str(n),' nu este numar prim!'])
end
```

Un al doilea exemplu verifică dacă un număr întreg, generat aleator între 3 şi 100, este par sau impar, respectiv dacă este număr prim.

```
%% par_impar.m
% Verifica paritatea sau imparitatea unui numar intreg generat aleator
% intre 3 si 100
% Verifica daca numarul este prim sau nu.
%% date intrare
```

```
n=randi([3,100]);
% se utilizeaza functia remainder si isprime
if rem(n,2)>0
    if isprime(n)
        disp([num2str(n),': impar si prim'])
    else
       disp([num2str(n),': impar si nu e numar prim'])
    end
else
    disp([num2str(n),': numar par'])
end
```

Se observă că se pot utiliza mai multe blocuri if imbricate. Este foarte important să se indenteze codul, pentru a urmări cursul aplicaţiei, cum se închide fiecare structură.

Forma cea mai generală a unui bloc if conţine mai multe condiţii, precedate de elseif, având următoarea sintaxă:

if <conditie 1>
    bloc 1
elseif <conditie 2>
    bloc 2
……
elseif <conditie k>
    bloc k
else
    bloc k+1
end

Pentru o astfel de structură se evaluează prima condiţie, iar dacă aceasta este adevărată se va executa blocul 1 de instrucţiuni. Dacă prima condiţie nu este adevărată, se evaluează următoarea condiţie, deci blocul 2 se va executa în situaţia când prima condiţie este falsă şi a doua adevărată.

În continuare se prezintă o secvenţă de cod, care calculează un impozit progresiv în funcţie de venit:

```
if Venit <= 2500 % 15% impozit
 Impozit = Venit * 0.15
elseif Venit <= 5000 %28% impozit
 Impozit = 375 + ((Venit - 2500) * 0.28)
elseif Venit <= 6000  %31% impozit
 Impozit = 1075 + ((Venit - 5000) * 0.31)
elseif Venit <= 7000 %36% impozit
 Impozit = 1385 + ((Venit - 6000) * 0.36)
else %40% impozit
 Impozit = 1745 + ((Venit - 7000) * 0.4)
end
```

La construirea acestor structuri este importantă ordinea în care se plasează condiţiile. În cazul când în exemplul precedent s-ar fi inversat ordinea condiţiilor, toţi plătitorii ar fi fost încadraţi în categoria 36%, deoarece prima condiţie (Venit < 7000) ar fi fost îndeplinită şi de plătitorii de 31%, 28% şi 15%.

Pentru a evita astfel de situaţii se recomandă utilizarea blocului **switch**. Acest tip de structură se recomandă în situaţia când blocurile de cod ce trebuie executate depind de valoarea unei singure variabile sau expresii. Această structură lucrează cu o singură expresie de test, ce

se evaluează la începutul structurii. Forma generală a unei astfel de condiţii de testare este:

```
switch expresie
  case expresie1
       bloc 1
  case { expr1, expr2, expr3,...}
       bloc 2
     ...
   otherwise
       bloc 3
end
```

Se observă că se poate face comparaţia cu o singură valoare sau cu o listă de mai multe valori (bloc 2). Dacă nici una din condiţii nu este îndeplinită se va executa blocul precedat de otherwise. Această secvenţă de cod este opţională, ea poate lipsi la fel ca varianta else din structura if.

Ex.

```
numar = input('Enter a number:');
switch numar
   case -1
      disp('numar unitar negativ');
   case 0
      disp('zero');
   case 1
      disp('numar unitar pozitiv');
   otherwise
      disp('alta valoare');
end
```

Pentru a construi diferitele condiţii ce intervin în structuri de tip if sau switch se utilizează expresii logice, evaluate cu valoarea true sau false. Expresiile logice se construiesc cu ajutorul operatorilor logici:

& - şi logic
| - sau logic
~ - Nu logic
== - egal
~= - diferit

Operatorii logici se pot aplica în cazul matricelor de aceeaşi dimensiune, lucrând element cu element, sau între un scalar şi o matrice, lucrând între scalar şi fiecare element al matricei. Ex. Dacă:

A = [0 1 1 0 1];
B = [1 1 0 0 1];
A & B = [0 1 0 0 1];
A | B = [1 1 1 0 1] ;
~A = [1 0 0 1 0];

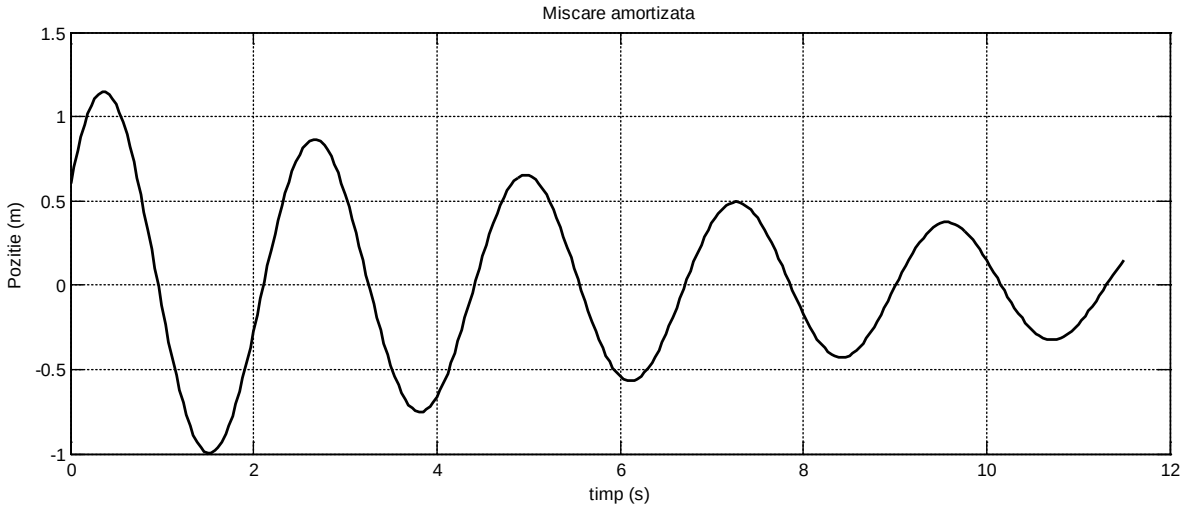
Obs. Se pot înlocui operatorii logici cu funcţiile: and(A, B), or(A, B), not(A) sau xor(A, B) – funcţia sau exclusiv

xor(A,B) = [1 0 1 0 0]

In numeroase programe sunt necesare variabile de control, care au valori logice. Se recomandă atribuirea de nume specifice, de exemplu introducerea unui prefix is, care semnalizează valoarea logică. In exemplul următor se reprezintă grafic mişcarea amortizată sau

nu, a unei mase legată de un arc. In cod există variabila isAmortizare, care modifică funcţia corespunzător. Reprezentarea grafică se observă în fig. 3.1. a cu amortizare şi b mişcare neamortizată.

```
%%oscilatii.m
% reprezinta grafic miscare oscilatorie masa – arc
%% date intrare
T=2.3; % periada miscarii (s)
ymax=1.2; %amplitudinea (m)
faza=pi/3; %faza (rad)
Tmax=5*T; % durata miscarii
isAmortizare=true;
tau=8.2; % constanta de timp pt amortizare
%% calculare functie
t=linspace(0,Tmax,300);
y=ymax*cos(2*pi*(t/T)-faza);
if isAmortizare
    y=y.*exp(-t/tau);
end
%%reprezentare grafica
plot(t,y);
xlabel('timp (s)');
ylabel('Pozitie (m)')
if isAmortizare
    title('Miscare amortizata')
else
    title('Miscare neamortizata')
end
grid on
```



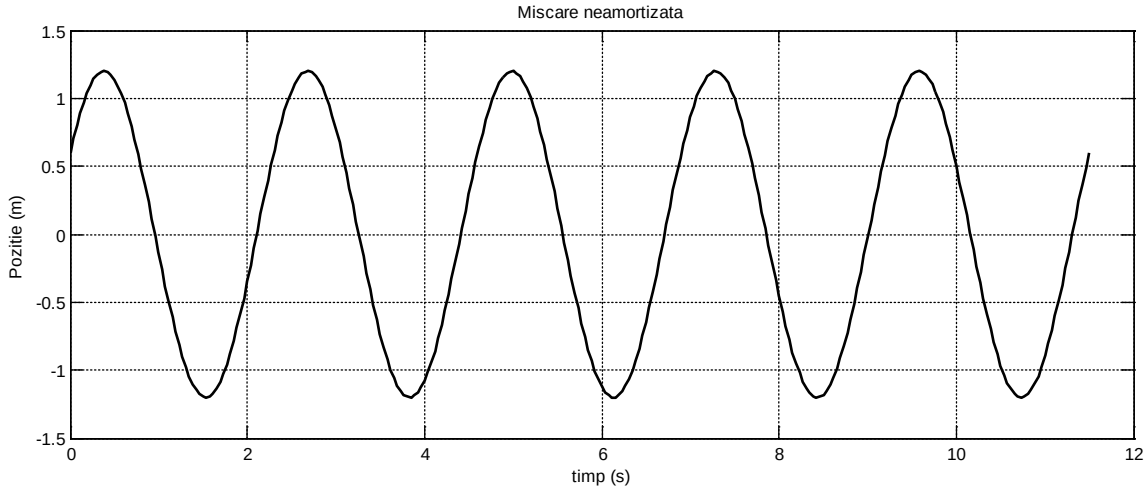


Fig. 3.1 Mişcare oscilatorie cu şi fără amortizare

Instrucţiuni repetitive

În Matlab sunt definite o serie de bucle, ce pot controla evoluţia unei aplicaţii.

Bucla **for – end** permite repetarea de un număr predeterminat de ori a unui grup de comenzi. Sintaxa este:

```
for variabila=vector
    comenzi
end
```

Comenzile din interiorul buclei se vor executa pentru fiecare element al vectorului. Când aplicaţia ajunge la comanda end, se modifică valoarea variabilei la următoarea valoare a vectorului şi se reia bucla, până în momentul când ajunge la ultimul element al vectorului. Pot exista mai multe bucle for cuprinse una în alta. Din punct de vedere al eficienţei programării, se recomandă evitarea buclei for, dacă acest lucru este posibil.

```
for i=1:100;
    xi=sin(i*pi/100);
end
```

se recomandă să fie scrisă:

```
i=1:100;
x=sin(i*pi/100);
```

Viteza de lucru în al doilea caz este mai mare. În primul caz, Matlab-ul lucrează cu alocare dinamică de memorie, adică la fiecare iteraţie se modifică dimensiunea vectorului x. Pentru a fi mai eficientă bucla, dimensiunea vectorului x ar trebui declarată în prealabil, cu o instrucţiune de forma: **x=zeros(1,100);**. Prin vectorizarea algoritmului (cazul doi) timpul de execuţie scade foarte mult.

În bucla for nu este permisă reatribuirea de valori variabilei de contorizare. La sfârşitul buclei, variabila de contorizare are ultima valoare egală cu ultima valoare a vectorului. Se admite folosirea separatorului : pentru definirea vectorului:

```
for index=val_initiala:increment:val_finala
```

Ex. Se prezintă în continuare o aplicaţie care desenează steluţe în interiorul unei bucle, apărând astfel senzaţia de mişcare.

```
%%desen_stea.m
%deseneaza stele in miscare
```

```
%% date intrare
nb=32; %nr spatii deplasare
dt=0.005; % timp pauza
ns=4; %nr spatii
```

```
%% bucla afisare
for ks=1:ns
    for kb=1:nb
        disp([blanks(kb),'*']);
        pause(dt)
    end
    for kb=nb:-1:1
        disp([blanks(kb),'*'])
        pause(dt)
    end
end
```

Programul utilizează comanda blanks(n) care returnează un şir cu n spaţii goale.
Buclele for se pot imbrica, dar trebuie să fie cuprinse unele în altele, să nu se intersecteze

Bucla while. Spre deosebire de bucla for, ce se parcurge de un număr predeterminat de ori, bucla while se parcurge în funcţie de valoarea logică a unei condiţii. Sintaxa este:

while conditie
    comenzi
end

Bucla se va parcurge atâta timp cât condiţia este adevărată. În momentul când programul ajunge la sfârşitul buclei, se reevaluează condiţia. Este obligatoriu ca pe parcursul buclei să se modifice valorile variabilelor ce intervin în condiţie, deoarece altfel bucla devine infinită.

3.2 Animaţii

Modelarea comportamentului unor elemente este analizată în multe situaţii prin asocierea ei cu animaţia mişcării elementelor.

Simularea mişcării se poate face în Matlab printr-o serie de reprezentări grafice cu mici diferenţe între ele, ce se afişează rapid pentru a creea iluzia mişcării. Acest lucru se realizează prin repetarea comenzii plot urmată de comanda drawnow (face actualizarea reprezentării), de obicei într-o buclă for.

Animaţii simple

Pentru a simula mişcarea unui obiect punctiform de-a lungul unei traiectorii, obiectul se poate reprezenta printr-un cerc. Fiecare poziţie intermediară a obiectului de-a lungul traiectoriei se stochează într-o matrice, iar cu ajutorul unei bucle for se redesenează obiectul în fiecare instanţă de timp.

In exemplul următor se realizează mişcarea unui obiect punctiform de-a lungul unei drepte orizontale, din x = 0 până la x = 1, iar coordonata y este păstrată constantă y = 0.3.

```
%% anim_simpla.m
%deplasarea unei particule de-a lungul unei drepte orizontale
%% setare parametri
xmin=0;
xmax=1;
```

```
y=0.3;
n=100; % nr intervale timp
%% stocare pozitii
x=linspace(xmin,xmax,n);
%% reprezentare
for i=1:n
    plot(x(i),y,'ro');
    axis([0 1 0 1]);
    drawnow
end
```

La fiecare iteraţie din bucla for se parcurg etapele:

- comanda plot şterge reprezentarea anterioară şi desenează o nouă reprezentare. In exemplul prezentat se desenează doar simbolul cerc, cu culoare roşie;
- comanda axis setează scalarea graficului. Este important să apară această comandă deoarece astfel se suprimă opţiunea de autoscalare care poate să distorsioneze reprezentarea;
- comanda drawnow este necesară din motive tehnice, ea forţează actualizarea reprezentării grafice. Fără ea se revedea doar ultima reprezentare.

Pentru a vedea efectul comenzii axis, respectiv drawnow se poate încerca rularea programului transformând în comentariu aceste comenzi, pe rând.

In majoritatea cazurilor, animaţia impune existenţa a trei elemente într-o buclă for:

- o comandă/comenzi ce desenează o anumită secvenţă;
- comanda axis pentru a evita autoscalarea;
- comanda drawnow pentru actualizarea graficului la fiecare iteraţie.

In continuare se introduc în aplicaţia precedentă două elemente suplimentare: se uneşte cu o linie poziţia iniţială a obiectului cu poziţia curentă şi se preia controlul vitezei de mişcare prin introducerea unei comenzi pause, al cărei argument este timpul exprimat în secunde.

```
%% anim_simpla1.m
%deplasarea unei particule de-a lungul unei drepte orizontale
%trasare linie pozitie initiala - poz curenta
%% setare parametri
xmin=0;
xmax=1;
y=0.3;
n=100; % nr intervale timp
dt=0.03; % interval timp pentru intarziere
%% stocare pozitii
x=linspace(xmin,xmax,n);
%% reprezentare
for i=1:n
    plot([x(1) x(i)], [y y],'r',x(i),y,'ro');
    axis([0 1 0 1]);
    pause(dt)
    drawnow
end
```

Pentru a simula mişcarea unui punct pe un cerc de rază R, se porneşte de la ecuaţiile parametrice ale mişcării circulare:

$t \in [0,1]$

$x(t) = Rcos(2\pi t/T).$

$y(t) = Rsin(2\pi t/T)$

Pentru a realiza modelarea se construieşte discretizarea timplului, iar coordonatele poziţiei se vor stoca în variabilele vectoriale x şi y.

```
%% misc_circulara.m
% deplasarea unui punct pe un cerc
%% date intrare
R=1; %raza cerc
T=1; %perioada miscarii
dt=0.03; %increment de timp pt viteza de deplasare
n=100; % numar pasi
%% calcule
t=linspace(0,T,n);
x=R*cos(2*pi*t/T);
y=R*sin(2*pi*t/T);
%% reprezentare
for i=1:n
    plot(x(1:i),y(1:i),'k',x(i),y(i),'ko');
    axis(1.2*[-R R -R R]);
    axis square
    pause(dt)
end
```

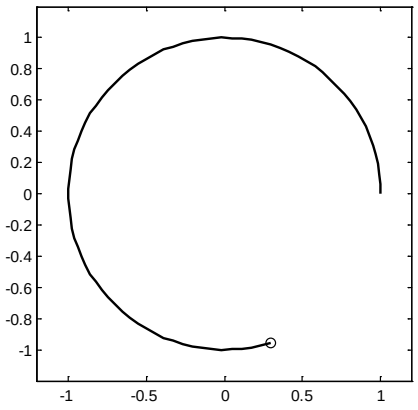


Fig. 3.2. Mişcarea circulară la un moment de timp

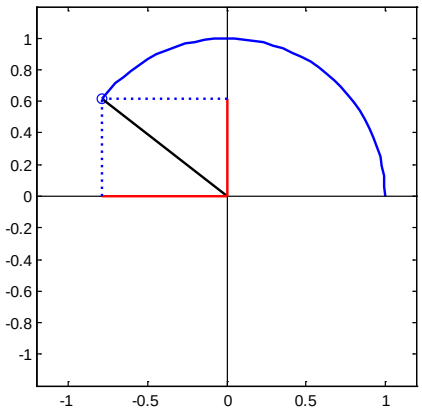


Fig. 3.3. Mişcarea circulară cu proiecţii

O fază a reprezentării se poate vedea în fig.3.2. Aplicaţia se poate completa prin desenarea unor linii suplimentare: axa 0x, axa 0y, raza, proiecţiile punctului pe axa 0x şi 0y şi verticala, respectiv orizontala din poziţia curentă pe axe (fig. 3.3).

```
%% misc_circulara1.m
% deplasarea unui punct pe un cerc
%% date intrare
R=1; %raza cerc
T=1; %perioada miscarii
dt=0.03; %increment de timp pt viteza de deplasare
n=100; % numar pasi
%% calcule
t=linspace(0,T,n);
x=R*cos(2*pi*t/T);
y=R*sin(2*pi*t/T);
%% reprezentare
for i=1:n
```

```
    %line([-R R],[0 0]);
    plot(x(1:i),y(1:i),'b',x(i),y(i),'bo',...
        1.2*[-R R],[0,0],'k',...
        [0 0],1.2*[-R R],'k',...
        [0 x(i)],[0 y(i)],'k',...
        [0 x(i)],[0 0],'r',...
        [0 0],[0 y(i)],'r',...
        [x(i) x(i)],[0,y(i)],'b:',...
        [0 x(i)],[y(i) y(i)],'b:');
    axis(1.2*[-1 1 -1 1]);
    axis square
    pause(dt)
end
```

În anumite situaţii este util să se realizeze şi animaţii pentru evoluţia unei funcţii în timp, de exemplu mişcarea unei unde sinusoidale cu lungimea de undă λ şi perioada T. Funcţia are expresia:

$$y(x,t) = A\sin(kx - \omega t), \tag{3.1}$$

unde k este numărul de undă (numărul de lungimi de undă cuprins pe distanţa 2π), iar ω este viteza unghiulară.

Pentru a realiza animaţia se generează două matrice, ce stochează spaţiul şi respective timpul. Pentru fiecare valoare de timp, întreaga curbă este calculate şi desenată ca o funcţie y(x).

```
%%anim_sin.m
% animatie functie sinus
```

```
%%setare parametri%% anim_sin.m
nx=200;
nt=200;
lambda=1; % lungime de unda
T=1; %perioada
xmax=3*lambda;
Tmax=4*T;
tpauza=0.02;
k=2*pi/lambda; %nr de unda
omega=2*pi/T; %viteza unghiulara
```

```
%% calcule si reprezentare
x=linspace(0,xmax,nx);
t=linspace(0,Tmax,nt);
for i=1:nt
    y=sin(k*x-omega*t(i));
    plot(x,y)
    xlabel('x')
    ylabel('y')
    pause(tpauza)
end
```

Curba este desenată pentru fiecare valoare a lui t, deci se face autoscalare şi nu mai este necesară comanda axis.

Următorul exemplu prezintă desenarea unei elipse.

```
%% anim_elipsa.m
```

```
%animeaza miscare pe elipsa
%x(t)=a*cos(omega*t+phi)
% y(t)=b*sin(omega*t-phi)
%% setare parametri
a=1.5;
b=1.25;
T=1; %perioada miscarii
np=4; % numar de perioade pt reprezentare
nt=400; %discretizare timp
phi=pi/3; % defazaj
%% calcule
tf=np*T; %valoare finala pt timp
omega=2*pi/T;
t=linspace(0,tf,nt);
x=zeros(1,nt);%initializare poz x
y=zeros(1,nt); %initializare poz y
x=a*cos(omega*t+phi);
y=b*sin(omega*t-phi);
xmin=min(x);
xmax=max(x);
ymin=min(y);
ymax=max(y);
%% reprezentare grafica
for i=1:nt
    plot(x(1:i),y(1:i),...
        x(i),y(i),'ro',...
        [0 x(i)],[0,y(i)],'r')
    axis(1.2*[xmin xmax ymin ymax]);
    axis square
    drawnow
end
```

Tehnicile de animaţie sunt foarte utile pentru înţelegerea descrierii matematice a mişcării corpurilor fizice în spaţiu, adică a cinematicii. Pentru simplificare se consideră mişcarea obiectelor punctiforme, deci se neglijează mişcarea de rotaţie şi extensia spaţială a obiectelor. În aceste condiţii mişcarea unui punct se poate descrie print trei entităţi: poziţie, viteză şi acceleraţie.

Fie poziţia unui punct x, iar viteza acestuia v. Între spaţiu şi viteză există relaţia:

$$v = \frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t)-x(t)}{\Delta t}. \quad (3.2)$$

Pentru Δt suficient e mic, se poate aproxima viteza pe baza diferenţei finite:

$$v \approx \frac{x(t+\Delta t)-x(t)}{\Delta t}. \quad (3.3)$$

Din relaţia 3.3 se poate estima spaţiul la un anumit moment de timp, pe baza poziţiei anterioare şi a vitezei corespunzătoare:

$$x(t+\Delta t) = x(t) + v \cdot \Delta t = x(i+1) = x(i) + v \cdot \Delta t. \quad (3.4)$$

a. Mişcarea cu viteză uniformă

Trebuie specificate datele de intrare: poziţia iniţială, viteza iniţială (constantă în acest caz), timpul pentru simulare şi incrementul de timp utilizat, Δt. Se iniţializează vectorii pentru stocarea timpului, spaţiului şi vitezei. În acest caz particular, mişcarea cu viteză uniformă,

viteza este constantă, deci vectorul pentru stocarea vitezei nu este necesar, dar pentru generalizare se menţine şi acest vector. În etapa următoare se calculează parametrii mişcării (fiind vorba de mişcare cu viteză uniform, e vorba doar de spaţiu), iar în final se face reprezentarea grafică.

```
%%miscare1D.m
%miscare cu viteza uniforma
%% setare date de intrare
v0=0.2; %m/s
x0=0; %m
tf=15; %s
deltat=0.05; %incrementul de timp s
%%initializarea variabile
t=0:deltat:tf;
n=length(t);% determinare dimensiune variabile
x=zeros(1,n);
v=zeros(1,n);
x(1)=0;
v(1)=v0;
%%calcule
for i=2:n
    v(i)=v(i-1);
    x(i)=x(i-1)+v(i)*deltat;
end
%% desenare
xmax=max(x);
xmin=min(x);
for i=1:n
    plot(x(i),0,'ro');
    axis([xmin, xmax, -1, 1]);
    xlabel('x(m)');
    drawnow
end
```

Foarte uşor se poate modifica programul pentru a reprezenta mişcarea unui punct într-un plan. Relaţiile corespunzătoare direcţiei x se aplică şi direcţiei y, iar iniţializarea se face şi pentru variabilele asociate direcţiei y. În cazul scalării graficului se determină valoarea minimă, respectivă maximă pentru fiecare axă, iar minimul şi maximul absolut se aplică în comanda axis, pentru a avea aceeaşi scară pe ambele axe.

```
%%miscare2D_unif.m
%miscare 2D cu viteza uniforma
%% setare date de intrare
v0x=0.15; %m/s
v0y=0.1; %m/s
x0=0; %m
y0=0; %m
tf=15; %s
deltat=0.05; %incrementul de timp s
```

```
%%initializarea variabile
t=0:deltat:tf;
n=length(t);% determinare dimensiune variabile
x=zeros(1,n);y=zeros(1,n);
vx=zeros(1,n);vy=zeros(1,n);
x(1)=0;y(1)=0;
vx(1)=v0x;vy(1)=v0y;
%%calcule
for i=2:n
    vx(i)=vx(i-1);vy(i)=vy(i-1);
    x(i)=x(i-1)+vx(i)*deltat;
    y(i)=y(i-1)+vy(i)*deltat;
end
%% desenare
maxim=max(max(x),max(y));
minim=min(min(x),min(y));
for i=1:n
    plot([x(1),x(i)],[y(1),y(i)],x(i),y(i),'ro');
    axis([minim, maxim, minim, maxim]);
    xlabel('x(m)');ylabel('y(m)');
    drawnow
end
```

Aplicaţia se poate extinde cu introducerea unei condiţii care să limiteze deplasarea pe o anumită lungime L, adică se simulează ciocnirea elastică cu un perete. În acest caz, la fiecare ciocnire trebuie să se schimbe sensul vitezei.

```
%%miscare1D_ciocnire.m
%miscare cu viteza uniforma intre 0 si L m
%% setare date de intrare
v0=0.2; %m/s
x0=0; %m
xf=1; %m
tf=15; %s
deltat=0.05; %incrementul de timp s
L=0.6;%m
%%initializarea variabile
t=0:deltat:tf;
n=length(t);% determinare dimensiune variabile
x=zeros(1,n);
v=zeros(1,n);
x(1)=0;
v(1)=v0;
%%calcule
for i=2:n
    v(i)=v(i-1);
    x(i)=x(i-1)+v(i)*deltat;
    if x(i)>L
```

```
        v(i)=-v(i);
    elseif x(i)<0
        v(i)=-v(i);
    end
end
%% desenare
xmax=max(x);
xmin=min(x);
for i=1:n
    plot(x(i),0,'ro');
    axis([xmin, xmax, -1, 1]);
    xlabel('x(m)');
    drawnow
end
```

Prin modificarea aplicaţiei anterioare la două dimensiuni şi limitarea spaţiului accesibil pe ambele direcţii se poate simula mişcarea unei bile de biliard.

```
%%biliard.m
%miscare cu viteza uniforma in spatiu delimitat pe ambele axe
%% setare date de intrare
v0x=0.3; %m/s
v0y=0.1; %m/s
x0=0; %m
y0=0;
xf=0.9; %m
yf=0.5; %m
tf=15; %s
deltat=0.05; %incrementul de timp s

%%initializarea variabile
t=0:deltat:tf;
n=length(t);% determinare dimensiune variabile
x=zeros(1,n);y=x;
vx=zeros(1,n);vy=vx;
x(1)=0;y(1)=0;
vx(1)=v0x;vy(1)=v0y;
%%calcule
for i=2:n
    vx(i)=vx(i-1);vy(i)=vy(i-1);
    x(i)=x(i-1)+vx(i)*deltat;
    y(i)=y(i-1)+vy(i)*deltat;
    if x(i)>xf
        vx(i)=-vx(i);
    elseif x(i)<0
        vx(i)=-vx(i);
    end
```

```
    if y(i)>yf
        vy(i)=-vy(i);
    elseif y(i)<0
        vy(i)=-vy(i);
    end
end
%% desenare
for i=1:n
    plot(x(i),y(i),'ro',x(1:i),y(1:i),'k');
    axis([0, xf, 0, yf]);
    xlabel('x(m)');ylabel('y(m)')
    drawnow
end
```

b. Mişcarea uniform accelerată

În cazul mişcării uniform accelerate de-a lungul axei 0x există relaţiile:

$$\begin{cases} a = \frac{dv}{dt} \\ v = \frac{dx}{dt} \end{cases} \quad (3.5)$$

Similar cu exemplele anterioare se pot scrie relaţiile:

$$v(t + \Delta t) = v(t) + a \cdot \Delta t, \quad (3.6)$$

$$x(t + \Delta t) = x(t) + v \cdot \Delta t. \quad (3.7)$$

Se pune problema valorii vitezei v, ce apare în relaţia (3.7), deoarece viteza este variabilă, există o valoare la momentul t şi o altă valoare la momentul t+Δt. Dacă se utilizează valoarea vitezei corespunzătoare momentului t, v(t), se obţine o aproximare mai puţin bună, cunoscută sub numele de metoda lui Euler. Pentru a îmbunătăţi aproximarea se recomandă utilizarea valorii medii a vitezei pe intervalul respectiv. În aceste condiţii se poate scrie:

$$x(t + \Delta t) = x(t) + \frac{1}{2}[v(t + \Delta t) + v(t)]\Delta t \quad (3.8)$$

Cu formulele (3.6) şi (3.8) se pot calcula parametri mişcării.

```
%%misc2D_a_ct.m
% miscare uniform accelerata 2D
%% setare date intrare
x0=0; %m
y0=0;
v0x=1; %m/s
v0y=10;
a0x=0; %m/s^2
a0y=-9.81;
tf=5; %s
deltat=.05; %s
%% initializare variabile
t=0:deltat:tf;
n=length(t);
x=zeros(1,n);y=x;
vx=x;vy=x;
ax=a0x;ay=a0y;
```

```
x(1)=x0;y(1)=y0;
vx(1)=v0x;vy(1)=v0y;
%% calcule
for i=2:n
    vx(i)=vx(i-1)+ax*deltat;
    vy(i)=vy(i-1)+ay*deltat;
    x(i)=x(i-1)+(vx(i)+vx(i-1))/2*deltat;
    y(i)=y(i-1)+(vy(i)+vy(i-1))/2*deltat;
    if y(i)<0
        vy(i)=-vy(i);
    end
end
%% desenare
ymax=max(y);xmax=max(x);
for i=1:n
    plot(x(i),y(i),'ro',x(1:i),y(1:i),'k')
    axis([0, xmax, 0, ymax])
    xlabel('x(m)');ylabel('y(m)')
    pause(0.05)
    drawnow
end
```

c. Mişcare sub acţiunea unei forţe variabile

În numeroase situaţii practice se întâlneşte mişcarea unui corp sub acţiunea unei forţe variabile, de exemplu cazul unui corp solidarizat cu un arc încastrat la capătul opus. Dacă arcul are constanta elastică k şi corpul are masa m, conform legii lui Hooke, forţa elastică ce acţionează asupra corpului este:

$$F = -kx$$

Ţinând cont că x se modifică, rezultă o forţă variabilă. Cu metoda prezentată anterior se poate calcula viteza şi spaţiul în fiecare moment de timp, pe baza forţei de la începutul intervalului.

$$v(t+\Delta t) = v(t) + \frac{F(t)}{m}\Delta t \qquad (3.9)$$

$$x(t+\Delta t) = x(t) + \frac{1}{2}\big(v(t+\Delta t) + v(t)\big)\Delta t \qquad (3,10)$$

Pentru a obţine o aproximare mai bună pentru viteze, în relaţia (3.9), ar trebui aplicată forţa medie pe interval, dar forţa depinde de valoarea finală a lui x. O variantă ce poate fi aplicată este metoda Verlet, utilizată în dinamica moleculară. Ideea acestei metode constă în parcurgerea următorilor paşi:

- Forţa la momentul iniţial se utilizează la calcularea vitezei la mijlocul intervalului;
- Viteza la mijlocul intervalului se utilizează la calcularea spaţiului la sfârşitul intervalului;
- Cu noua poziţie, se calculează forţa la sfârşitul pasului;
- Forţa finală se utilizează pentru calcularea vitezei finale, plecând de la viteza la mijlocul intervalului.

Algoritmul metodei este:

Pas 1: $v_{1/2} = v(t) + \left[\frac{F(v(t),x(t),t)}{m}\right]\cdot\frac{\Delta t}{2}$

Pas 2: $x(t+\Delta t) = x(t) + v_{1/2}\Delta t$

Pas 3: $F_n = F\big(v_{1/2}, x(t+\Delta t), t+\Delta t\big)\Delta t$

Pas 4: $v(t+\Delta t) = v_{1/2} + \left[\frac{F_n}{m}\right]\frac{\Delta t}{2}$

```
%%miscare_F_var.m
% miscare punct sub actiunea fortei elastice
%% date intrare
tf=10; %s
x0=1; %m
v0=0; %m/s
a0=0; %m/s^2
k=0.5; %constanta elastica a arcului
m=0.1; %kg
deltat=.05;
%% initializare variabile
t=0:deltat:tf;
n=length(t);
x=zeros(1,n);
v=zeros(1,n);
F=zeros(1,n);
x(1)=x0;
v(1)=v0;
F(1)=-k*x(1);
%% calcule
for i=2:n
    v_jum=v(i-1)-k*x(i-1)/m*deltat/2;
    x(i)=x(i-1)+v_jum*deltat;
    Fn=-x(i)*k;
    v(i)=v_jum+Fn/m*deltat/2;
    F(i)=Fn;
end
%% desenare
xmin=min(x);
xmax=max(x)
for i=1:n
    plot(x(i),0,'ro')
    axis([xmin, xmax, -1, 1])
    pause(0.1)
    drawnow
end
```

Probleme

1. Să se scrie o aplicaţie (anim_cerc.m) care să animeze desenarea unui cerc definit parametric.

$r = r_0$

$x = r\cos(\omega t)$

$y = r\sin(\omega t)$

Să se traseze o linie, care uneşte centrul cercului cu fiecare punct în parte şi funcţia desenată până la momentul respectiv.

2. Să se scrie o aplicaţie care desenează un cerc ce se micşorează în timp, caracterizat prin

perioada T şi constanta de amortizare τ. De asemenea se desenează o linie, care uneşte centrul cercului cu punctul curent şi funcţia desenată până la momentul respectiv.

$r = r_0 e^{-\frac{1}{\tau}}$

$x = r(t)\cos(\omega t)$

$y = r(t)\sin(\omega t)$

3. Să se scrie o aplicaţie care să simuleze plimbarea aleatoare într-o singură dimensiune. Se pleacă din punctul x = 0 şi se face un pas la dreapta sau la stânga în funcţie de o generare aleatoare de numere (r = rand). Dacă r ≤ 0.5 se face un pas la dreapta (se adaugă o unitate la valoarea anterioară), dacă r > 0.5, se face un pas la stânga (se scade o unitate). Se va calcula matricea x, unde se stochează poziţia şi în final se face reprezentarea. La fiecare pas se avansează pe verticală.
4. Să se scrie o aplicaţie care să deseneze traiectoria descrisă de un lanţ cinematic format din două elemente articulate. Primul element de lungime R are o cuplă cinematică de rotaţie fixă, iar al doilea element, articulat de primul, are lungimea r. Elementele au vitezele unghiulare omega1, respectiv omega2.

$x(t) = Rcos(\omega_1 t) + rcos(\omega_2 t)$

$y(t) = R\sin(\omega_2 t) + rsin(\omega_2 t)$

Se vor parcurge 5 perioade complete la primul element, iar al doilea elemet va avea perioada definită în raport cu primul. (T_1/T_2= raport). Se începe cu R=5, r=2, raport = 9. Ce se întâmplă dacă r>R?

5. Un obiect aruncat pe verticală cu viteza iniţială v0 va avea la momentul t înălţimea h dată de relaţia

$h(t) = h_0 + v_0 t - \frac{1}{2} g t^2$,

unde h_0 este înălţimea iniţială, iar g =9.81 m/s^2 este acceleraţia gravitaţională.Corpul are energie potenţială (Eg) şi cinetică (Ec), respectiv energie totală.

$E_g = mgh$

$E_c = \frac{1}{2} m v^2$

$E = E_g + E_c$

m fiind masa obiectului şi v viteza acestuia. Să se scrie o aplicaţie miscare_verticala.m pentru o minge de tenis de 60 g, lovită la o înălţime de 0.8 m cu o viteză de 18 m/s pe o durată de 4 s. Se va calcula înălţimea h a mingii, viteza acesteia, energia potenţială, energia cinetică şi energia totală. Energiile se reprezintă în acelaşi grafic, iar toate trei reprezentările se vor pune în aceeaşi pagină cu comanda subplot.

3.3. Funcţii definite de utilizator

În programare funcţiile au un rol important, ele oferă un grad de generalitate şi permit aplicaţiilor complexe să fie segmentate în părţi mai simple, ce pot fi înţelese,scrise şi testate mai uşor.

Funcţiile au date de intrare şi ieşire. Intrările sunt valorile variabilelor trimise spre funcţie. Pot fi mai multe date de intrare şi ele pot fi de diferite clase: numere reale, vectori, matrice, şiruri etc. Pot exista situaţii când o funcţie are ca date de intrare o altă funcţie. În mod similar, ieşirile pot aparţine diferitelor clase.

În afară de calcularea valorilor de ieşire, o funcţie poate avea şi alte efecte. De exemplu, funcţia disp() – afişează în fereastra de comandă, plot() – generează o reprezentare grafică sau beep() care generează un sunet. Funcţiile pot avea orice număr de intrări, chiar 0, orice

număr de ieşiri şi orice număr de alte efecte. Funcţiile bine concepute au marele avantaj că pot fi reutilizate.

MATLAB are foarte multe funcţii predefinite: sin, cos, log, exp etc. Se pot defini funcţii de către utilizator care să extindă funcţiile existente. Dacă într-o anumită aplicaţie trebuie evaluată frecvent următoarea expresie matematică:

$f(x) = 2x^2 + 5x - 0.7$

se poate construi o funcţie în acest scop. Codul corespunzător este prezentat în continuare

```
function y=myfun(x)
% y=myfun(x) returneaza valoarea y(x)=2x^2+5x-0.7
a=2;
b=5;
c=-0.7;
y=a*x^2+b*x+c;
```

Funcţía se salvează sub numele myfun.m. Totdeauna trebuie ca numele fişierului să fie identic cu numele funcţiei, declarat în prima linie de cod. După salvare, funcţia poate fi apelată în fereastra de comandă, cu condiţia să fie salvată în directorul curent.

`myfun(1)` → 6.3

```
x=linspace(-1,1,100);
```

`plot(x,myfun(x))` → reprezentarea grafică

`help myfun` → afişază comentariul ce apare în codul funcţiei, după linia de declarare

Declararea funcţiei trebuie să fie făcută pe prima linie şi are forma:

```
function variabile iesire=nume_functie(argumente)
```

Totdeauna prima linie începe cu cuvântul cheie function. Variabilele de ieşire, din partea stângă a semnului =, declară numele variabilei (variabilelor) căreia i se atribuie valori de către funcţie.

```
function Tc=fc(Tf)
% returneaza temperatura in grade Celsius data de Tf in grade Fahrenheit
```

Argumentele specificate între paranteze sunt datele de intrare, se mai numesc parametri funcţiei. O funcţie poate apela altă funcţie. Ex. Există un program principal unde apare codul

```
x1=3;
a=4;
b=1;
x=mediegeom(a,b);
disp x
```

```
function y=mediegeom(x1, x2);
% calculeaza media geometrica pentru x1 si x2
a=sqrt(x1*x2);
y=a;
```

În linia 4 a programului principal se apelează funcţia mediegeom, trecând argumentele a şi b spre funcţie. Deci, la apelarea funcţiei, x1 va avea valoarea 4 (a), iar x2 valoarea 1 (b). Funcţia nu va modifica variabilele a sau b din programul de bază. În prima linie de cod a funcţiei se declară variabila de returnare y, ceea ce înseamnă că funcţia va atribui o valoare acestei variabile şi o va returna programului de bază. Valoarea lui y, în acest caz, va fi 2 şi va fi transmis programului de bază, unde se stochează în variabila x.

Fiecare funcţie are un spaţiu de lucru pentru variabile, care este distinct de spaţiul de lucru principal. În exemplul precedent există variabila x1 şi în programul de bază, şi în funcţie, dar

ele nu se influenţează reciproc. Acest lucru reprezintă un avantaj major pentru că la scrierea unei funcţii nu trebuie avut în vedere numele variabilelor utilizate în afara acesteia. Funcţia interacţionează cu restul codului, doar prin datele de ieşire şi intrare. Acest lucru permite reutilizarea unei funcţii definite de utilizator.

O funcţie poate avea orice număr de intrări şi ieşiri. Datele de intrare sunt specificate ca argumente ale funcţiei (ex. mediegeom(x1, x2)), iar datele de ieşire sunt specificate în linia de declarare a funcţiei. Dacă sunt mai multe, ele se specifică între paranteze drepte .

```
function [a,b]=myfunc(x,y,z)
```

După linia de declarare a funcîiei se adaugă un comentariu explicativ pentru funcţie, care este returnat de MATLAB la tastare `help nume_functie`.

Varianta cea mai generală şi accesibilă de scriere a unei funcţii este într-un fişier separat, ce are numele identic cu al funcţiei. Există, însă, şi alte moduri de organizare ce pot fi utile uneori.

Dacă se defineşte o funcţie ce realizează o sarcină complexă, ea poate fi segmentată în mai multe subfuncţii simple, care se pot salva în acelaşi fişier. Astfel, în loc de un director cu mai multe fişiere mici, există un singur fişier, ce conţine toate funcţiile. Evident, aceste subfuncţii nu mai pot fi apelate din alte programe.

Fie o funcţie medii.m

```
function [ma,mg,marm]=medii(a,b)
% calculeaza media aritmetica, geometrica si armonica pentru numerele a si b
ma=mean(a,b);
mg=mediegeom(a,b);
marm=mediearm(a,b);
%-----------------------------------------------------
function y=mediegeom(x1, x2);
% calculeaza media geometrica pentru x1 si x2
a=sqrt(x1*x2);
y=a;
%-----------------------------------------------------
function y=mediearm(a,b);
% calculeaza media armonica pentru x1 si x2
 a=2/(1/x1+1/x2);
 y=a;
------------------------------
```

Se pot întâlni şi funcţii imbricate, cu condiţia să nu se intersecteze.

```
function f1=myfunc1(a,b,c)
……….
      function f2=myfunc2(x)
      ……
      end
end
```

Dacă se lucrează cu funcţii imbricate, obligatoriu codul funcţiei se încheie cu `end`. Avantajul oferit de acest mod de lucru este faptul că spaţiul de lucru al funcţiei din exterior este moştenit de funcţia din interior, deci nu mai trebuie atribuite valori. Dar, trebuie menţionat, că existenţa acestei organizări, îngreunează mult depanarea aplicaţiei, motiv pentru care nu se recomandă utilizarea acestei organizări, decât în cazuri speciale.

O altă categorie de funcţii sunt funcţiile anonime, scrise pe o singură linie de cod, ce pot fi definite fără scrierea unui fişier. Ele permit evitarea existenţei mai multor fişiere individuale.

Sintaxa pentru definirea unei funcţii cu numele afunc, având o variabilă de intrare x este:

```
afunc=@(x) expresie
```

Partea din stânga semnului =, este o entitate care lucrează similar cu numele funcţiei. Această atribuire creează o variabilă de tip function handle cu numele afunc. Declararea variabilei se face cu construcţia `@(x)`, ceea ce înseamnă că variabila funcţiei este x, iar expresie trebuie să fie o singură relaţie, ce evaluează valoarea funcţiei în raport cu x. Variabila x este locală, fiind legată doar de definirea funcţiei, astfel încât nu intră în conflict cu altă variabilă cu acelaşi nume, din altă parte a programului. Atât argumentul funcţiei, cât şi valoarea returnată pot fi de tip matricial. Se pot defini funcţii anonime şi de mai multe variabile.

```
sinc2=@(x,y) sin(sqrt(x.^2+y.^2))/sqrt(x.^2+y.^2)
```

O funcţie poate fi transmisă ca argument pentru altă funcţie în două moduri:

- prin pointerul către funcţie (function handle);
- ca o funcţie anonimă.

Pointerul către o funcţie include caracterul @ înaintea numelui funcţiei, fie că este vorba de o funcţie predefinită, fie că este o funcţie definită utilizator: @sin, @exp sau @medii. In cazul funcţiilor anonime, la definirea funcţiei, membrul stâng este chiar pointerul către aceasta. Pentru exemplul anterior se poate apela direct afunc, dacă se doreşte ca această funcţie să fie transmisă către o altă funcţie.

Function handle este o noţíune care permite adresarea funcţiei din memorie, permiţând transferarea unei funcţii ca argument pentru o altă funcţie sau la modul mai general permite tratarea funcţiei ca un tip de date. Dacă există o funcţie definită cu numele myfun, ea poate fi apelată prin apelarea @myfun.

În următorul exemplu, funcţia myDesenare primeşte ca prim argument o astfel de funcţie, iar ea poate fi apelată ca şi cum ar fi numele funcţiei.

```
function myPrincipal
figure(1)
myDesenare(@sin,-2*pi,2*pi)
figure(2)
myDesenare(@cos,-2*pi,2*pi)
```

```
function myDesenare(fun,a,b)
% deseneaza functia fun intre a si b
n=200;
x=linspace(a,b,n);
y=fun(x);
plot(x,y)
```

Acest tip de construcţie a codului este utilă pentru a lucra cu funcţii, ce preiau alte funcţii ca argument.

Probleme

1. Să se scrie o funcţie c2f, care face conversia temperaturii din grade Celsius în grade Fahrenheit.

Relaţia de legătură este: $Tc = (Tf - 32) * \frac{5}{9}$

2. Să se scrie o funcţie (invers) care să inverseze ordinea elementelor unui vector sau unui şir de caractere

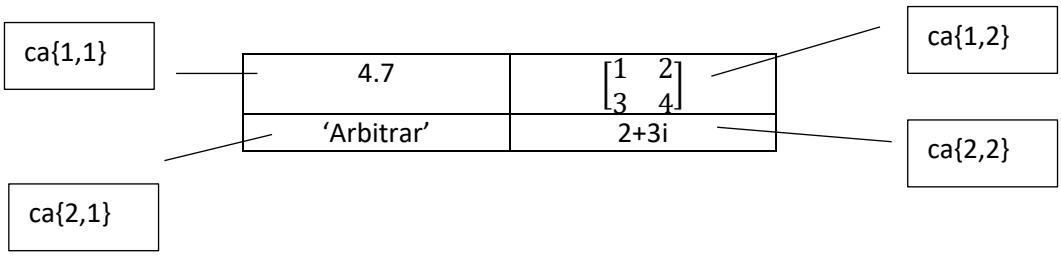
3. Să se scrie o funcţie (polar2rect) care returnează coordonatele carteziene (x, y) corespunzătoare coordonatelor polare (r, θ).
4. Să se scrie un program test_c2f.m care să utilizeze funcţia c2f pentru a calcula temperatura în grade Celsius corespunzătoare temperaturii Fahrenheit în domeniul 100 până la 150, din 2 în 2 grade.
5. Să se scrie o funcţie care să limiteze, la valori impuse, superior şi inferior elementele unui vector. Elementele care depăşesc valoarea maximă se egalează cu ea, iar cele care sunt sub limita inferioară, devin egale cu aceasta.
6. Scrieţi o funcţie (swap) care să inverseze două elemente între ele (a să devină b şi b să devină a).
7. Scrieţi o funcţie (bubbleSort) care să facă sortarea elementelor unui vector pe baza metodei bubble sort (se examinează pe rând fiecare element şi se compară cu vecinul din dreapta, dacă se respectă relaţia de ordonare, elementele rămân neschimbate, dacă nu se inversează).
8. Scrieţi o funcţie (bubbleSortVizual) care să creeze o reprezentare vizuală a metodei precedente. Aplicaţia se va face pentru vector de dimensiune mică (ex. 10 elemente).
9. Scrieţi o funcţie (vrotit) care să rotească circular cu k spaţii un vector sau un şir. Pentru k pozitiv deplasarea se face spre dreapta, iar pentru k negativ, deplasarea se face spre stânga.
10. Scrieţi o funcţie anonimă care să evalueze funcţia

$f(x, \delta) = \frac{1}{1+e^{x/\delta}}$

Funcţia se va reprezenta grafic pentru x€[-1, 1] şi δ =[0.001, 0.01, 0.05, 0.1, 0.2, 0.4]

Clase şi structuri de date

Datele de tip cell array (matrice complexă) permit combinarea mai multor tipuri de date într-un vector sau o matrice cu două dimensiuni, entităţile fiind indexate după numărul de linii şi coloane. Natura heterogenă a elementelor este o caracteristică unică. De ex. O matrice 2 x 2 poate conţine în celule un număr, o matrice, un şir şi un număr complex.



Pentru indexarea elementelor se utilizează { }.Pentru a genera o astfel de variabilă se scrie codul

```
ca=cell(2,2);
ca{1,1} = 4.7;
 ca{1,2} = [1, 2; 3, 4];
ca{2,1}='Arbitrar';
ca{2,2}=2+3i;
```

Un anumit element particular din matricea de celule poate fi accesat şi utilizând şi al doilea nivel de indexare, dacă acest lucru este posibil.

Ex. disp(ca{1,1}) → 4.7

disp(ca{1,2} (2,2)) → 4

disp(ca{2,1} (3:5)) → 'bit'

Acest tip de date permite stocarea şirurilor de lungimi diferite.

Declararea datelor de tip cell array se face cu comanda cell(dimensiune) sau cell(dim1, dim2…dimk).

Matricea şi matricea de tip celule combină date individuale în obiecte compozite, iar elementele individuale sunt accesate pe baza unui sau mai multor indici. Uneori este util să se combine diferite tipuri de date şi să fie accesate pe bază de nume, nu pe bază de indice.

Clasa struct din MATLAB creează astfel de entităţi şi permite ca specificarea acestora să se facă pe bază de nume. Clasa deţine un număr de câmpuri, care permit stocarea informaţiei în acest mod. De ex. Informaţia legată de o anumită carte poate include autorul, anul publicaţiei, titlul, număr de pagini etc. La variabilele de tip struct, câmpurile sunt specificate prin separare cu punct.

```
Carte.titlu='Analiza si procesarea datelor'
Carte.autor='Arjana Davidescu'
Carte.an=2001
Carte.nrpag=256
```

În acest caz câmpurile sunt: titlu, autor, an şi nrpag. Pentru a extrage o anumită informaţie, de ex. anul publicaţiei, se apelează

```
disp(['Anul publicatiei este ', num2str(Carte.an)]
```

În mod similar cu alte clase MATLAB, şi clasa struct poate fi combinată într-o matrice. De ex. mai multe cărţi pot fi combinate într-o matrice

```
biblioteca(1)=Carte;
biblioteca(2).titlu='Mecanisme';
biblioteca(2).autor='Carmen Sticlaru';
biblioteca(3).titlu='Actionari hidraulice'
```

Definirea unei astfel de variabile se poate face cu sintaxa

```
Nume_var=struct('nume_camp','valoare'…)
```

Pentru exemplul anterior

```
carte=struct('titlu','Analiza si procesarea datelor','autor', 'Arjana Davidescu','an','2001');
```

O alta posibilitate este

```
data.x=linspace(0,2*pi,30);
data.y=sin(data.x);
data.titlu='Functie sin';
plot(data.x,data.y),title(data.titlu)
```

Probleme

1. Fie lista

Nume	Prenume	Medie
Ionescu	Gabriel	8,75
Georgescu	Vasile	8,25
Dumitriu	Ioan	9,20
Popescu	Anca	9,02

Să se scrie un program ce construieşte un vector de tip struct având câmpurile Nume, Prenume, Medie. Să se tipărească o anumită înregistrare (o linie din tabel) în funcţie de valoarea indicată de utilizator

2. Să se scrie o funcţie cuvinte=words_ca(s) ce returnează o variabilă cell array, ce conţine cuvintele dintr-un şir s. Cuvintele se identifică prin spaţiul liber. Se poate utiliza o funcţie predefinită din MATLAB, strtrim ce elimină spaţiile libere dintr-un şir.
3. Să se scrie o funcţie sout=cuv_invers(s), ce returnează cuvintele din şirul s, dar în ordine inversă.