Document Test - Summary Programare evolutiva si algoritmi genetici

Programare evolutiva si algoritmi genetici (Academia de Studii Economice din Bucure□ti)

Generarea unei permutari

```
%generare permutare
%i: n -dimensiune
%E: p - permutarep p

function p = permutare(n)

   p = zeros(1,n);
   for i = 1:n
      gata = 0;
      while ~ gata
         x= unidrnd(n);
         if ~ ismember(x,p)
            p(i)=x;
            gata=1;
         end;
      end;
   end;
```

function [v] = evaluare(p)

```
%evaluare permutare (asezare pe tabla)
%I:p-permutare
%E:v-valoarea functiei obiectiv
[~,n]=size(p);
v=0;
for i=1:n-1
   for j=i+1:n
      if abs(i-j)==abs(p(i)-p(j))   %am gasit o pereche care se ataca
         v=v+1;
      end
   end
end
v=n*(n-1)/2 -v;

end
```

Generarea populatiei :

1.Generare populatie prin reprezentarea binara

```
function [pop]=gen_ini_binar(m,dim)
% genereaza populatie cu indivizi reprezentati ca siruri binare
% I: dim - dimensiune populatie, m - dimensiune individ
% E: pop - populatia
   pop=zeros(dim,m);
```

```
   for i=1:dim
      x=unidrnd(2^m-1);
      pop(i,1:m)=bitget(x,m:-1:1);
   end;
end
```

2. **Generarea populatiei cu reprezentare prin numere intregi**

```
function [] = pop_reprez_intregi(dim,n,pm,M,N)
pop = zeros(dim,n);
for i =1:dim
   for k=1:n
      pop(i,k)=unidrnd(N-M+1)+M-1;
   end;
end;
disp('pop in reprez cu nr intregi');
disp(pop);
%aplic mutatia pe populatia pop
popN=mutatie_intregi_ra(pop,pm,M,N);
end
```

3. **Generarea unei populatii prin permutari**

```
function y=gen_perm(m)
% genereaza permutare
%I: m - dimensiune permutare
%E: y - permutare
   y=zeros(1,m);
   for i=1:m
      gata=0;
      while(~gata)
         x=unidrnd(m);
         if(~ismember(x,y))
            y(i)=x;
            gata=1;
         end;
      end;
   end;
end
```

4.Generarea unei populatii prin reprezentare in nr reale

```
function Pop=gen_pop_perm(dim,m)
%generarea populatiei initiale de permutari pe multimea m
%I: dim - nr. indivizi, m - dimensiune individ (permutare)
%E: pop - populatia
```

```
  Pop=zeros(dim,m);
  for i=1:dim
    Pop(i,1:m)=gen_perm(m);
  end;
end
```

Mutatii

1.Operatorul de mutatie prin amestec

```
function y=m_amestec(x,pm)
  % mutatie prin amestec pentru reprezentare cu permutari
  % I: x - individul (permutare); pm - probabilitate de mutatie
  % E: y - individ mutat
  [~,m]=size(x);
  y=x;
  r=unifrnd(0,1);
  if r<pm
    p1=unidrnd(m);
    p2=p1;
    while p2==p1
      p2=unidrnd(m);
    end;
    a=gen_perm(p2-p1+1);
    y(p1:p2)=x(p1-1+a(1:p2-p1+1));
  end;
end
```

sau

```
function [y] = m_amestec(x,pm)
 %operatorul de mutatie prin amestec
 %I:x-individ,pm-probabilitatea
 %e:y-individul rezultat
 dim=length(x);
 y=x;
 for i=1:dim
   r=unifrnd(0,1);
   if r<pm
    p=unidrnd(dim,1,2);
    while p(1)==p(2)
      p(2)=unidrnd(dim)
    end;
    p1=min(p);
    p2=max(p);
    %disp([modifica intre'num2str(p1) 'si'num2str(p2)']);
```

```
      a=permutare(p2-p1+1);
       for i=p1:p2
            y(i) = x(p1-1+a(i-p1+1));
       end;
    end;
 end;

end
```

sau

```
function [ popN ] = mutatie_amestec( pop, pm )
%MUTATIE_AMESTEC
[dim,n]=size(pop);
popN=pop;
for i=1:dim
   r=unifrnd(0,1);
   if(r<pm)
      disp('Mutatie efectuata in cromozomul'); disp(pop(i,:));
      p=zeros(1,2);
      p(1)=unidrnd(n);
      p(2)=unidrnd(n);
      while(p(1)==p(2))
         p(2)=unidrnd(n);
      end;
      poz=sort(p);
      disp('Pozitiile:');
      disp(poz);
      x=zeros(dim, n);
      x=unidrnd(poz(2)-poz(1),poz(1), poz(2))-1;
      unidrnd(poz(1), poz(2), 1, n);
      disp('Cromozom rezultat');disp(popN(i,:));
   end;
end;
```

2.Operatorul de mutatie prin amestec intre 2 pozitii

```
function [PopM] = mutatie_amestec(pm,dim,m)
% operatorul de mutatie prin amestec intre doua pozitii
% I: pm - probabilitate de mutatie; dim - dim. populatie; m - nr. cromozomi
% E: populatie mutata
   pop=gen_pop_perm(dim,m);
   PopM=pop;
   for ind=1:dim
```

```
      r=unifrnd(0,1);
      if(r<pm)
         disp(['Mutatie in cromozomul ' num2str(ind)]);
         disp(PopM(ind,:));
         i=0;j=0;
         while(i==j)
            poz=unidrnd(m,1,2);
            i=min(poz);
            j=max(poz);
         end;
         disp(['Pozitiile ' num2str(i) ' si ' num2str(j)]);
         p_amestec=gen_perm(j-i+1);
         disp('Amestecul este conform permutarii de pozitii:');
         disp(p_amestec+i-1);
         PopM(ind,i:j)=PopM(ind,p_amestec(1:j-i+1)+i-1);
         disp('Cromozomul rezultat');
         disp(PopM(ind,:));
      end;
   end;
end
```

3.Operatorul de mutatie prin interschimbare

```
function [y]=m_interschimbare(x,pm)
   % operatorul de mutatie prin interschimbare pe permutari
   % I: x - individul (permutarea), pm - probabilitatea de mutatie
   % E: y - individul modificat

   y=x;
   r=unifrnd(0,1);
   if r<pm
      n=length(x);
      poz=unidrnd(n,1,2);
      while poz(1)==poz(2)
         poz(2)=unidrnd(n);
      end;
      poz=sort(poz);
      y(poz(1))=x(poz(2));
      y(poz(2))=x(poz(1));
   end;
end
```

4.Operatorul de mutatie prin inversiune

```
function [y]=m_inversiune(x,pm)
   % operatorul de mutatie prin inversiune pe permutari
```

```
  % I: x - individul (permutarea), pm - probabilitatea de mutatie
  % E: y - individul modificat

  y=x;
  r=unifrnd(0,1);
  if r<pm
    n=length(x);
    poz=unidrnd(n,1,2);
    while poz(1)==poz(2)
      poz(2)=unidrnd(n);
    end;
    poz=sort(poz);
    y(poz(1):poz(2))=x(poz(2):-1:poz(1));
  end;
end
```

sau

```
function [ popN ] = mutatie_inversiune( pop, pm )
%MUTATIE_INVERSIUNE presupune selectarea aleatoare a douã gene i si j
%ºi inversarea ordinii în secvenþa dintre cele douã poziþii.
[dim,n]=size(pop);
popN=pop;
for i=1:dim
  r=unifrnd(0,1);
  if(r<pm)
    disp('Mutatie efectuata in cromozomul'); disp(pop(i,:));
    p=zeros(1,2);
    p(1)=unidrnd(n);
    p(2)=unidrnd(n);
    while(p(1)==p(2))
      p(2)=unidrnd(n);
    end;
    poz=sort(p);
    disp('Pozitiile:');
    disp(poz);
    c=0;
    for j=poz(1):poz(2)

      popN(i, j)=pop(i,pop(i,poz(2))-c);
      c=c+1;
    end;
```

5.Operatorul de mutatie prin inserare

```
function [ popN ] = mutatie_inserare( pop, pm )
%MUTATIE_INSERARE permutarea care sufera o mutatie
[dim,n]=size(pop);
popN=pop;
for i=1:dim
   r=unifrnd(0,1);
   if(r<pm)
      disp('Mutatie efectuata in cromozomul'); disp(pop(i,:));
      p=zeros(1,2);
      p(1)=unidrnd(n);
      p(2)=unidrnd(n);
      while(p(1)==p(2)&&p(2)==p(1)+1)
         p(2)=unidrnd(n);
      end;
      poz=sort(p);
      disp('Pozitiile:');
      disp(poz);
      popN(i,1:poz(1))=pop(i,1:poz(1));
      popN(i,poz(1)+1)=pop(i,poz(2));
      popN(i,poz(1)+2:poz(2))=pop(i,poz(1)+1:poz(2)-1);
      popN(i,poz(2)+1:n)=pop(i,poz(2)+1:n);
      disp('Cromozom rezultat');
   end;
end;
```

6. Operatorul de mutatie pentru reprezentarea prin nr intregi

```
function [popN]=mutatie_intregi_ra(pop,pm,M,N)
[dim,n]=size(pop);
popN=pop;
for i=1:dim
   efectuat=0;
   for k=1:n
      r=unifrnd(0,1);
      if(r<pm)
         disp('Mutatie efectuata in cromozomul');
         disp(pop(i,:));
         disp('Gena');
         disp(k);
         % este generat aleator un numar intreg intre M si N:
         % genereaza aleator un numar R intre 1 si N-M+1: unidrnd(N-M+1)
         % aduna la R valoarea M-1
         popN(i,k)=M-1+unidrnd(N-M+1);
         efectuat=1;
      end;
   end;
```

```
  if(efectuat)
    disp('Cromozom rezultat');
    disp(popN(i,:));
  end;
end;
end
```

7. Operatorul de mutatie uniforma pt reprezentarea cu siruri de numere reale

```
function [ y ] = m_uniforma( x,pm,a,b )
%operator de mutatie uniforma
%I: individul asupra caruia se aplica mutatia x, prob de mutatie pm,
%capetele de interval a,b
%E: individul obtinut y

[~,n]=size(x);
y=x;
for i=1:n
  r=unifrnd(0,1); % generam o valoare aleatoare intre 0 si 1
  if r<pm % verificam daca e mai mica decat prob de mutatie
    y(i)=unifrnd(a,b);
  end;
end;

end
```

8. Generarea op de mutatie in reprez cu siruri reale(mutatia neuniforma)

```
function [ popNoua ] = mutatie_neunif_reala( pop, pm, a, b, sigma)
%MUTATIE_NEUNIF_REALA Summary of this function goes here
%   E:populatia noua
%pop=gen_pop_nr_reale(a,b,dim);
[dim, n]=size(pop);
popNoua=pop;
for i=1:dim
  c=0;
  for j=1:n
    r=unifrnd(0,1);
    if(r<pm)
      x=normrnd(0, sigma);%x=generat normal cu medie0 si
      %deviatie sigma
      popNoua(i, j)=pop(i, j)+x;
      if(popNoua(i,j)<a)
        popNoua(i,j)=a;
      else
        if(popNoua(i,j)>b)
          popNoua(i,j)=b;
        end;
```

```
        end;
        c=1;
      end;
    end;
if(c)
   disp('populatia dupa mutatie:')
   disp(popNoua(i,:));
end;
end;

end
```

9. Operatorul de mutatie reprezentarea binara

```
function [pop_m]=mutatie_binar(m,dim,pm)
   pop=genereaza_ini_binar(m,dim);
   pop_m=pop;
   V=[];
   %aplica operatorul de mutatie in reprezentarea binara
   for i=1:dim
      for j=1:m
         r=unifrnd(0,1);
         if(r<pm)
            pop_m(i,j)=not(pop(i,j));
            if(~ismember(i,V))
               V=[V;i];
            end;
         end;
      end;
   end;
   disp('Populatia pe care se aplica mutatia');
   disp(pop);
   disp('Indicii indivizilor mutanti:');
   disp(V);
   disp('Populatia mutanta');
   disp(pop_m);
end
```

10. Operatorul de mutatie de tip fluaj in reprezentarea cu nr intregi

```
function y=test_fluaj_i(pm,a,b,sigma)
   % mutatia fluaj pentru reprezentare cu numere intregi
   % I: x - individ; pm - probabilitatea de mutatie;
   %    a,b - dom. de definitie; sigma - deviatia pentru val. de fluaj
   %    sigma < t/3, unde t este pragul de modificare (fluaj)
   % E: y - individ mutat
   dim=10;
   x=gen_ini_nint(a,b,dim);
```

```
  [~,m]=size(x);
  y=x;
  for i=1:m
    r=unifrnd(0,1);
    if r<pm
      y(i)=x(i)+fix(normrnd(0,sigma));
      if y(i)<a
        y(i)=a;
      end;
      if y(i)>b
        y(i)=b;
      end;
    end;
  end;
end
```

11. Operatorul de mutatie de tip fluaj pentru reprezentarea cu nr reale

```
function [popN]=mutatie_fluaj_reale(pm,a,b,sigma,dim)
% este aplicata mutatia fluaj pe cromozomi din [a1,b1]x...x[an,bn]
pop=gen_ini_n_reale(a,b,dim);
[~,n]=size(a);
popN=pop;
for i=1:dim
  efectuat=0;
  for k=1:n
    r=unifrnd(0,1);
    if(r<pm)
      disp(['Mutatie efectuata in cromozomul ' num2str(i)]); disp(pop(i,:));
      disp(['Gena ' num2str(k)]);
      % R este generat aleator normal, cu medie 0 si deviatie sigma
      R=normrnd(0,sigma);
      popN(i,k)=pop(i,k)+R;
      if(popN(i,k)<a(k))
        popN(i,k)=a(k);
      else if (popN(i,k)>b(k))
          popN(i,k)=b(k);
        end;
      end;
      efectuat=1;
    end;
  end;
  if(efectuat)
    disp('Cromozom rezultat');disp(popN(i,:));
  end;
end;
end
```

12. **function [popN] = mutatie_permutare_interschimb(pop, pm)**

```
%MUTATIE_PERMUTARE_INTERSCHIMB
%E: populatia noua
[dim, n]=size(pop);
popN=pop;
for i=1:dim
   r=unifrnd(0,1);
   if(r<pm)
      %generez un vector de o linie si 2 coloane care reprezinta gena i
      %si j generate aleator
      p=zeros(1,2);
      p(1)=unidrnd(n);
      p(2)=unidrnd(n);
      while(p(1)==p(2))
         p(2)=unidrnd(n);
      end;
      poz=sort(p);
      popN(i,poz(1))=pop(i, poz(2));
      popN(i, poz(2))=pop(i, poz(1));
      disp(popN(i,:));
   end;
end;


end
```

Operatori de recombinare

1.Operatorul de recombinare multipunct

```
function [ y1,y2 ] = crossover_multipunct( x1,x2,pr,nr)
%operatorul de incrucisare multipunct
%I: indivizii parinti x,y;
%pr-prob de crossover
%nr-numarul de puncte de incrucisare
%E: copii/progenituri/descendenti a,b

y1=x1; y2=x2;
r=unifrnd(0,1);
if(r<pr)
   [~,m]=size(x1); %tilda ins ca ignoram primul rezultat
   poz=zeros(1,nr);
```

```
  for i=1:nr
    temp=unidrnd(m);
    while ismember(temp,poz)
      temp=unidrnd(m);
    end;
    poz(i)=temp;
  end;
  poz=sort(poz);
  poz=[poz,m]; %operator de concatenare, mai adaugam un element pe orizontala

  for i=1:2:nr
    y1(poz(i):poz(i+1))= x2(poz(i):poz(i+1));
    y2(poz(i):poz(i+1))= x1(poz(i):poz(i+1));
  end;
end;
```

2.Operatorul de recombinare uniforma

```
function [popN]=crossover_uniform(pc,dim,m,p)
% pop este populatia de parinti
pop=gen_pop_binar(m,dim);
disp('Populatia de parinti:');
disp(pop);
[dim,m]=size(pop);
popN=pop;

for k=1:2:dim-1
    x1=pop(k,1:m);
    y1=pop(k+1,1:m);
    r=unifrnd(0,1);
    disp('Parintii:');
    disp(x1);
    disp(y1);

    if(r<=pc)
        disp('Incrucisarea este realizata pe baza valorilor:');
        for i=1:m
           v=unifrnd(0,1);
           disp(v);
           if(v<=p)
               popN(k,i)=pop(k,i);
               popN(k+1,i)=pop(k+1,i);
           else
               popN(k,i)=pop(k+1,i);
               popN(k+1,i)=pop(k,i);
           end;
       end;
   else
       disp('Incrucisare asexuata');
   end;
   disp('Urmasii:');
   disp(popN(k,1:m));
   disp(popN(k+1,1:m));
```

```
end;
end
```

3. **Operatorul de recombinare unipunct**

```
function [y1,y2] = crossover_unipunct(x1,x2,poz)
   %I:x1,x2-parinti
   %I:poz-punctul de unde incepe incrucisarea genelor celor 2 parinti
   %E: y1,y2-copiii
   [m,~]=size(x1);
   y1=x1;
   y2=x2;
   y1(1:poz)=x1(1:poz);
   y1(poz+1:m)=x2(poz+1:m);
   y2(1:poz)=x2(1:poz);
   y2(poz+1:m)=x1(poz+1:m);
end
```

4. **Operatorul de recombinare aritmetica simpla pentru reprezentarea cu nr reale**

```
function [x2,y2]=rec_aritm_simpla(x1,y1,pr,w)
   %operator de recombinare aritmetica simpla
   %I:x1,y1-parintii
   %pr-probabilitate de recombinare
   %w-pondere
   %E: x2,y2-descendentii
   x2=x1;
   y2=y1;

   [~,m] = size(x1);
   for i=1:m
      if unifrnd(0,1) < pr
         x2(i) = y1(i)*w + x1(i)*(1-w);
         y2(i) = x1(i)*w + y1(i)*(1-w);

      end;
   end;
```

5. **Operatorul de recombinare aritmetica totala pentru reprezentarea cu nr reale**

```
function [ a,b ] = rec_aritm_tot(x,y,p,w)
%operator de combinare aritmetica totala
%i:parinti(x,y), probabilitatea(p) si pondera(w);
%e: copii(a,b)

a=x;
```

```
b=y;
[~,m] = size(x);
for i=1:m
   if unifrnd(0,1) < p
      a(i) = x(i)*w + y(i)*(1-w);
      b(i) = x(i)*(1-w) + y(i)*w;

   end;
end;


function [y] = n_fluaj_R(x,pm,a,b,disp)
 %operatul de fluaj prin reprezentarea prin numere realae
 %I:x-individ,cromozon,pm-prob de mutatie,a,b-capetele,interval,disp-disp
 %val de fluaj
 %e:y-individul rezultat
 dim=length(x);
 y=x;
 for i=1:dim
    R=unifrnd(0,1);
    if R < pm
     y(i)=x(i) + normrnd(0,disp);
     if (y(i)>b)
        y(i)=b;
     end;
     if y(i) < a
        y(i) = a;
     end;
    end;
 end;

end


function []=genpop_nrrealeunif(dim, n, pm, a, b, sigma)
% generarea populatiei care sufera mutatie este realizata
%aleator, uniforma pe [a,b]
pop=zeros(dim,n);
for i=1:dim
   for j=1:n
      pop(i,j)=unifrnd(a,b);
   end;
end;
disp('Populatia initiala');disp(pop);
%aplicarea mutatiei
popN=mutatie_neunif_reala(pop,pm,a,b,sigma);

end
```

```
function [Generatie]=s_gen_elitist(pop, desc)
   % selectarea elitista a generatiei noi
   % I: pop - populatia curenta, desc - descendentii generati
   %    pe ultima coloana se afla valoarea functiei obiectiv
   % E: Generatie - generatia noua

   Generatie=pop;
   [dim,n]=size(pop);
   [max1,i]=max(pop(:,n));
   [max2,j]=max(desc(:,n));
   if max1>max2
      [min1,k]=min(desc(:,n));
      Generatie(k,:)=pop(i,:);
   end;
end


function [parinti]=s_p_FPS_ruleta(pop)
   % selectia parintilor tip ruleta cu probabilitati FPS standard
   % I: pop - populatia curenta, pe ultima coloana e val. fct. obiectiv
   % E: parinti - parintii selectati

   [m,n]=size(pop);
   p=zeros(1,m);
   p(1:m)=pop(1:m,n);
   s=sum(p);
   p(1:m)=p(1:m)/s;
   q=zeros(1,m);
   for i=1:m
      q(i)=sum(p(1:i));
   end;
   parinti=zeros(m,n);
   for k=1:m
      r=unifrnd(0,1);
      i=1;
      while i<m && r>q(i)
         i=i+1;
      end;
      parinti(k,:)=pop(i,:);
   end;
   disp(p);
   disp(q);
end

function [parinti]=s_p_FPS_SUS(pop)
   % selectia parintilor tip SUS cu probabilitati FPS standard
```

```
    % I: pop - populatia curenta, pe ultima coloana e val. fct. obiectiv
    % E: parinti - parintii selectati

    [m,n]=size(pop);
    p=zeros(1,m);
    p(1:m)=pop(1:m,n);
    s=sum(p);
    p(1:m)=p(1:m)/s;
    q=zeros(1,m);
    for i=1:m
       q(i)=sum(p(1:i));
    end;
    parinti=zeros(m,n);
    i=1;
    k=1;
    r=unifrnd(0,1/m);
    while(k<=m)
       while(r<=q(i))
          parinti(k,1:n)=pop(i,1:n);
          r=r+1/m;
          k=k+1;
       end;
    i=i+1;
    end;
end

function [parinti]=s_p_turneu(pop,k)
    % selectia parintilor de tip turneu
    % I: pop - populatia curenta, k - numarul de participanti la turneu
    %    pe ultima coloana se afla valoarea functiei obiectiv
    % E: parinti - parintii selectati

    [m,n]=size(pop);
    parinti=zeros(m,n);
    turneu=zeros(k,n);
    for i=1:m
       for j=1:k
          t=unidrnd(m);
          turneu(j,:)=pop(t,:);
       end;
       [~,p]=max(turneu(:,n));
       parinti(i,:)=turneu(p,:);
    end;
end


function [a,b]=r_OCX(x,y,pr)
    % Operatorul de recombinare Order Crossover
```

```
    % I: x,y - cromozomii parinti, pr - probabilitatea de recombinare
    % E: a, b - descendenti

    a=x;
    b=y;
    r=unifrnd(0,1);
    if r<pr
        [~,m]=size(x);
        p=[0 0];
        p(1)=unidrnd(m);
        p(2)=unidrnd(m);
        while(p(1)==p(2))
            p(2)=unidrnd(m);
        end;
        p=sort(p);
        a=OCX(x,y,p);
        b=OCX(y,x,p);
    end;
end


function d=OCX(x,y,p)
    % Operatorul de incrucisare OCX, pentru un singur descendent
    % pentru al doilea descendent se apeleaza din nou, cu parintii y, x
    % I: x, y - cromozomii parinti, cele doua pozitii (in vectorul p)
    % E: d - primul cromozom descendent

    [~,m]=size(x);
    d=zeros(1,m);
    d(p(1):p(2)) = x(p(1):p(2));
    unde=p(2)+1;
    for i=[p(2)+1:m 1:p(2)]
        if ~ismember(y(i),d)
            if unde>m
                unde=1;
            end;
            d(unde)=y(i);
            unde=unde+1;
        end;
    end;
end


function [copil]=ocx_copil(x,y,p1,p2)
[~,n]=size(x);
copil=zeros(1,n);
copil(p1:p2)=x(p1:p2);
if(p2<n)
```

```
   poz_libera=p2+1;
else
   poz_libera=1;
end;
y_cautare=[y(p2:n) y(1:p2-1)];
for i=1:n
   if(~ismember(y_cautare(i),copil))
      copil(poz_libera)=y_cautare(i);
      if(poz_libera<n)
         poz_libera=poz_libera+1;
      else
         poz_libera=1;
      end;
   end;
end;
end
```

```
function [y]=m_interschimbare(x,pm)
   % operatorul de mutatie prin interschimbare pe permutari
   % I: x - individul (permutarea), pm - probabilitatea de mutatie
   % E: y - individul modificat

   y=x;
   r=unifrnd(0,1);
   if r<pm
      n=length(x);
      poz=unidrnd(n,1,2);
      while poz(1)==poz(2)
         poz(2)=unidrnd(n);
      end;
      poz=sort(poz);
      y(poz(1))=x(poz(2));
      y(poz(2))=x(poz(1));
   end;
end
```

```
function [Pop_urm] =elitism(Pop_curenta,Copii )
%presupunem ca dimensiunea populatiei de copii=dimensiunea populatiei la
%momentul curent
[dim,m]=size(Pop_curenta);
Pop_urm=Copii;
[val1,i1]=max(Pop_curenta(:,m));
[val2,~]=max(Copii(:,m));
if(val1>val2)
```

```
   best=Pop_curenta(i1,:);
   %este inlocuit un copil aleator
   ind=unidrnd(dim);
   Pop_urm(ind,:)=best;
end;
end



function [ v ] = evaluare( p )
%evaluare permutare (asezare pe tabla)
%I:p-permutare
%E:v-valoarea functiei obiectiv
[~,n]=size(p);
v=0;
for i=1:n-1
   for j=i+1:n
      if abs(i-j)==abs(p(i)-p(j))   %am gasit o pereche care se ataca
         v=v+1;
      end
   end
end
v=n*(n-1)/2 -v;

end

function [  ] = matrice( p )
%functia care arata tabla

[~,n]=size(p);
mat=zeros(n,n);
for i=1:n
   mat(i,p(i))=1;
end;
disp(mat);

end



function [ pop ] = populatie( m,n)
%generare populatie
%I:m-indivizi, n-dimensiune permutare
%E:pop-populatia  m*(n+1)

pop=zeros(m,n+1);
for i=1:m
   x=permutare(n);
   pop(i,1:n)=x;
```

```
   pop(i,n+1)=evaluare(x);
end;

end
```