Procesarea Cunoștințelor și Calcul Inteligent

Algoritmul Particle Swarm Optimization – partea a III-a

Adăugarea elementelor de afișare a rezultatelor. Testarea algoritmului

Pentru a se observa mai bine structura algoritmului, înainte de a transforma programul în unul iterativ, vom condensa structura codului, transformând pașii principali în funcții și reducând cantitatea de cod scris.

Calculul funcției de adaptare

Codul existent

```
% initializare vector functii de adaptare (coloana)
fadapt=zeros(np,1);

% calcul diferente
for i=1:np % pentru fiecare particula din populatie
    % extragere particula din populatie
    vact=pop(i,:);
    % calcul diferenta absoluta fata de vectorul de referinta
    df=sum(abs(vref-vact));
    % scriere diferenta in vectorul functiilor de adaptare
    fadapt(i)=df;
end;
```

Avem nevoie, ca variabile de intrare, de

- `np` - numărul de indivizi din populație, un număr întreg
- `pop` – populația curentă - o matrice
- `vref` – vectorul codurilor ASCII ale cuvântului de ghicit

La ieșire,

- `fadapt` – funcțiile de adaptare calculate pentru populație - vector coloană

Obținem funcția:

```
function [fadapt]=calcul_functie_adaptare(np,pop,vref)
% Calculul functiei de adaptare pentru PSO ghicire cuvant,
% diferenta absoluta dintre vectorul codurilor ASCII ale cuvantului
% gasit de algoritm si vectorul codurilor ASCII ale cuvantului de ghicit
%
% Date de intrare:
% np - numarul de particule din populatie
% pop - populatia curenta (matrice cu np linii)
% vref - vectorul numeric al codurilor ASCII ale cuvantului de ghicit
%
% Date de iesire:
% fadapt - functiile de adaptare ale particulelor din populatia curenta
%                 - vector coloana cu np elemente
```

```
% initializare vector functii de adaptare (coloana)
fadapt=zeros(np,1);

% calcul diferente
for i=1:np % pentru fiecare particula din populatie
    % extragere particula din populatie
    vact=pop(i,:);
    % calcul diferenta absoluta fata de vectorul de referinta
    df=sum(abs(vref-vact));
    % scriere diferenta in vectorul functiilor de adaptare
    fadapt(i)=df;
end;
```

Calculul pbest – gbest

Codul inițial:

```
% gaseste particulele care trebuie inlocuite
inloc=find(fadapt<fpbest);

% actualizeaza matricea pbest si vectorul fpbest
pbest(inloc,:)=pop(inloc,:);
fpbest(inloc,:)=fadapt(inloc,:);

% verifica daca s-a gast un noul lider gbest
lid=find(fpbest==min(fpbest));

% daca am gasit altul mai bun, inlocuiesc liderul
if fpbest(lid(1))<fgbest
    fgbest=fpbest(lid(1));
    gbest=pbest(lid(1),:);
end;
```

Variabile de intrare pentru funcție:

- `pop` - populația curentă
- `fadapt` – funcțiile de adaptare ale populației curente
- `pbest` - populația paralelă a particulelor personal best
- `fpbest` - funcțiile de adaptare ale populației `pbest`
- `gbest` – particula lider sau global best
- `fgbest` - funcția de adaptare a liderului

Variabile de ieșire:

- `pbest` - populația paralelă a particulelor personal best, actualizată
- `fpbest` - funcțiile de adaptare ale populației `pbest`, actualizată
- `gbest` – particula lider sau global best, actualizată
- `fgbest` - funcția de adaptare a liderului, actualizată

Obținem funcția:

```
function [pbest,fpbest,gbest,fgbest] = actualizare_best
(pop,fadapt,pbest,fpbest,gbest,fgbest)
 % Date de intrare:
% pop - populatia curenta, matrice
% fadapt - functiile de adaptare ale populatiei curente, vector coloana
% pbest - populatia paralela personal best pentru particule, matrice
% fpbest - functiile de adaptare ale populatiei pbest, vector coloana
% gbest - particula lider, vector linie
% fgbest - functia de adaptare a particulei lider, numar
%
% Date de iesire:
% matricele si vectorii pbest,fpbest,gbest,fgbest, dupa actualizare

 % gaseste particulele care trebuie inlocuite
inloc=find(fadapt<fpbest);

% actualizeaza matricea pbest si vectorul fpbest
pbest(inloc,:)=pop(inloc,:);
fpbest(inloc,:)=fadapt(inloc,:);

% verifica daca s-a gast un noul lider gbest
lid=find(fpbest==min(fpbest));

% daca am gasit altul mai bun, inlocuiesc liderul
if fpbest(lid(1))<fgbest
    fgbest=fpbest(lid(1));
    gbest=pbest(lid(1),:);
end;
```

Recalcularea vitezei și a poziției particulelor

Extragem doar prima și ultima parte a codului, fără validare pe care o vom transforma separat în funcție.

Codul initial:

```
   % extrage particula din populatie si viteza ei curenta
   p=pop(i,:);
   v=vit(i,:);

   % extrage particula pbest
   pbp=pbest(i,:);

   % vectorii random
   r1=rand(1,dp);
   r2=rand(1,dp);

   % recalculeaza viteza particulei
   v=v+2*r1.*(pbp-p)+2*r2.*(gbest-p);

   % recalculeaza pozitie particula
   p=p+v;
```

```
   % rotunjeste valori elemente particula la numere intregi
   p=round(p);
```

Condensăm codul astfel:

```
   % recalculeaza viteza particulei
v=vit(i,:)+2*rand(1,dp).*(pbest(i,:)-pop(i,:))+2*rand(1,dp).*(gbest-
pop(i,:));

% recalculeaza pozitie particula si rotunjeste valorile
p=round(pop(i,:)+v);
```

Validarea

```
   % validare
   for j=1:dp % pentru toate elementele particulei modificate
       % limita inferioara
       if p(j)<linf
           p(j)=linf;
       end;
       % limita superioara
       if p(j)>lsup
           p(j)=lsup;
       end;
   end;
```

Variabile de intrare:

- `dp` - lungimea unei particule
- `p` - patricula supusă validării
- `inf` - limita inferioară admisibilă pentru valorile elementelor unei particule
- `lsup` - limita inferioară admisibilă pentru valorile elementelor unei particule

Variabile de ieșire:

- `p` - particula după validare

Obținem funcția:

```
function [p]=validare_cuvant(dp,p,linf,lsup)
% Validarea unei particule cu valori iesite din limitele admisibile
% in algoritmul PSO ghicire cuvant
% valorile depasite sunt inlocuite cu valorile extreme admise
%
% Date de intrare:
% dp - lungimea unei particule
% p - patricula supusa validarii, vector linie
% linf - limita inferioara admisibila pentru valorile elementelor
%        unei particule, numar
% lsup - limita inferioara admisibila pentru valorile elementelor
%        unei particule, numar
% Date de iesire:
% p -  particula dupa validare
```

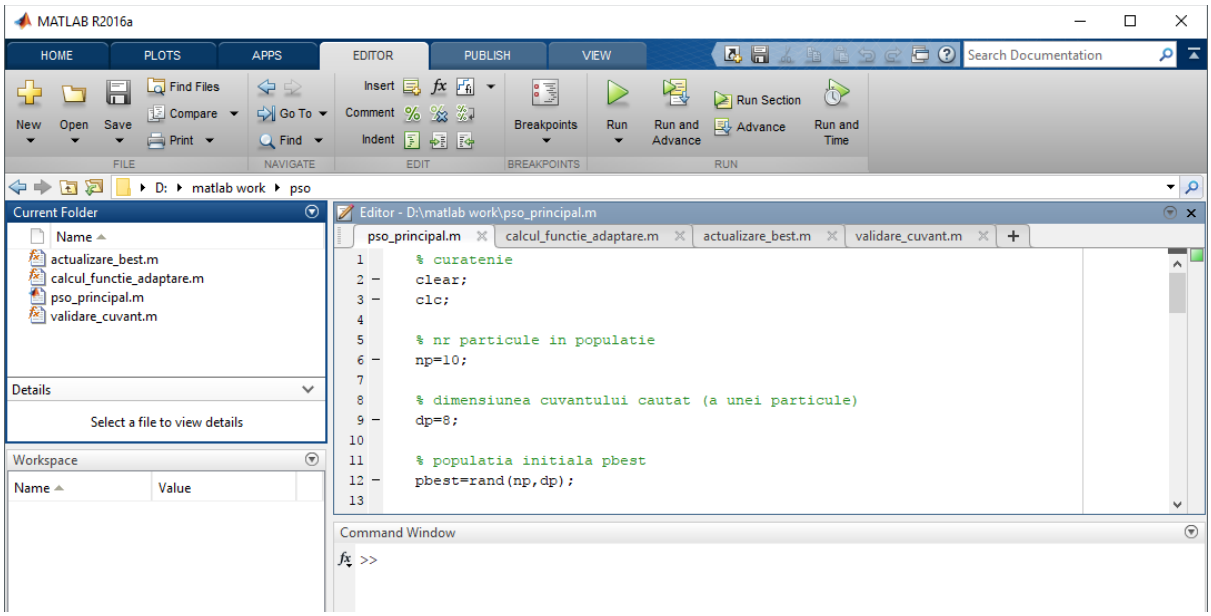
```
for j=1:dp % pentru toate elementele particulei modificate
    % limita inferioara
    if p(j)<linf
        p(j)=linf;
    end;
    % limita superioara
    if p(j)>lsup
        p(j)=lsup;
    end;
end;
```

În programul inițial, liniile transformate în funcții vor fi înlocuite cu apeluri către funcțiile respective. Vom obține:

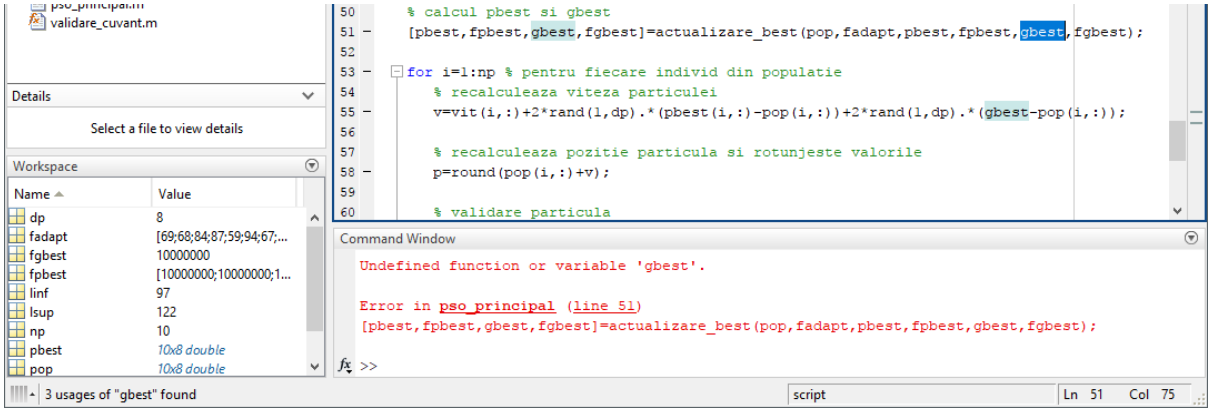
```
% curatenie
clear;
clc;

% nr particule in populatie
np=10;

% dimensiunea cuvantului cautat (a unei particule)
dp=8;

% populatia initiala pbest
pbest=rand(np,dp);
% functiile de adaptare ale populatiei pbest
fpbest=10e6*ones(np,1);

% functia de adaptare initiala a liderului
fgbest=10e6;

% vitezele initiale
vit=0.01*rand(np,dp);

% limite valori elemente individ
linf=97;
lsup=122;

 % generare populatie
%pop=round(linf+rand(np,dp)*(lsup-linf));

% populatia initiala pentru teste
pop=[99,121,98,100,98,102,98,113;
    104,111,119,113,116,103,114,110;
    117,110,120,109,110,119,98,121;
    98,103,117,116,109,98,99,113;
    120,109,99,115,120,109,110,117;
    115,113,104,120,112,101,99,108;
    109,114,105,119,112,121,117,108;
    111,107,114,105,118,115,117,118;
    103,106,100,114,117,110,115,99;
    108,122,115,102,111,109,101,100];

% vector numeric cuvant cautat (vector de referinta)
vref=double('albastru');
```

```
% calcul functie de adaptare
[fadapt]=calcul_functie_adaptare(np,pop,vref);

% calcul pbest si gbest
[pbest,fpbest,gbest,fgbest]=actualizare_best(pop,fadapt,pbest,fpbest,gbest,
fgbest);
 for i=1:np % pentru fiecare individ din populatie
   % recalculeaza viteza particulei
   v=vit(i,:)+2*rand(1,dp).*(pbest(i,:)-pop(i,:))+2*rand(1,dp).*(gbest-
pop(i,:));

   % recalculeaza pozitie particula si rotunjeste valorile
   p=round(pop(i,:)+v);

   % validare particula
   [p]=validare_cuvant(dp,p,linf,lsup);

  % inlocuieste particula veche din populatie cu particula noua
   % si viteza veche cu viteza noua
   pop(i,:)=p;
   vit(i,:)=v;
end;
```

Ca să nu amestecăm funcțiile algoritmului PSO cu alte programe, punem într-un director separat scriptul programului principal modificat (`pso_principal.m`) și cele trei funcții. Pentru ca programul să poată funcționa corect, funcțiile apelate trebuie să se găsească în același director cu programul care le apelează.



Rulând programul modificat, primim un mesaj de eroare:



La linia 51, indicăm particula `gbest` ca variabilă de intrare pentru o funcție, dar această particulă nu a fost inițializată, așadar nu există.

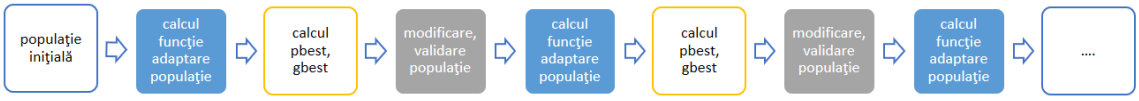
Ca să reparăm eroarea, în zona de inițializări, pe lângă funcția de adaptare, trebuie să inițializăm și liderul. Valoarea elementelor inițializate nu contează, putem folosi un vector cu valori 0.

```
% liderul initial al roiului
gbest=zeros(1,dp);

% functia de adaptare initiala a liderului
fgbest=10e6;
```

Acum, programul recalculează populația conform așteptărilor. Putem trece mai departe, la calculul iterativ.

Calculul iterativ

Conform algoritmului de principiu, calculul iterativ se desfășoară astfel:



Iterația începe cu calculul funcției de adaptare a populației curente și se termină cu actualizarea populației. În iterația următoare, se începe cu calculul funcției de adaptare a populației modificate ș.a.m.d.

Implementăm calculul iterativ folosind o buclă `for` (am păstrat doar comentariile, care arată pașii programului):

```
% curatenie
% initializari
% generare populatie
% vector numeric cuvant cautat (vector de referinta)

for iter=1:50
    % calcul functie de adaptare
    % calcul pbest si gbest
    for i=1:np % pentru fiecare individ din populatie
        % recalculeaza viteza particulei
```

```
        % recalculeaza pozitie particula si rotunjeste valorile
        % valideaza particula
        % inlocuieste particula veche din populatie cu particula noua
        % si viteza veche cu viteza noua
    end;
end;
```

Pentru a încheia algoritmul, mai realizăm următoarele modificări:

În procesul iterativ, adăugăm și un factor suplimentar de inerție (engl. *momentum*) aplicat vitezei. Inerția începe cu valori mici la început (pentru a nu frâna viteza de căutare în faza inițială) și crește treptat, ajungând la valoarea maximă către sfârșitul iterațiilor, pentru ca deplasarea să se facă pe distanțe cât mai scurte și să se exploreze cât mai bine spațiul unde, se presupune, se află soluția optimă. De fapt, în algoritm, viteza extrasă din matricea vitezelor se înmulțește cu un factor scalar subunitar, care ia valori între maximul recomandat 0.9 (la început, viteza este redusă puțin) și minimul recomandat 0.4 . Înmulțind succesiv viteza cu valori subunitare din ce în ce mai mici, către final, ea ar trebui să tindă spre 0.

Pentru a descrește uniform valoarea factorului de inerție pe parcursul iterațiilor, avem nevoie de numărul de iterații ca parametru definit în zona de inițializare

În program, inerția se modelează astfel. Între comentariile care marchează pașii programului, am inserat doar liniile de program adăugate sau modificate.

```
% curatenie
% initializari

% numarul de iteratii
nrit=50;

% factor initial de inertie
fi=0.9;
% rata de actualizare a factorului de inertie
dif=(0.9-0.4)/nrit;


% generare populatie
% vector numeric cuvant cautat (vector de referinta)

for iter=1:nrit
    % calcul functie de adaptare
    % calcul pbest si gbest
    for i=1:np % pentru fiecare individ din populatie
        % recalculeaza factor inertie
        fi=fi-dif;
        % recalculeaza viteza particulei
        v=fi*vit(i,:)+2*rand(1,dp).*(pbest(i,:)-
pop(i,:))+2*rand(1,dp).*(gbest-pop(i,:));
        % recalculeaza pozitie particula si rotunjeste valorile
        % valideaza particula
        % inlocuieste particula veche din populatie cu particula noua
        % si viteza veche cu viteza noua
    end;
end;
```

În sfârșit, adăugăm câteva elemente de afișare. Dorim să vedem, după fiecare iterație, soluția optimă găsită de algoritm și să reprezentăm grafic descreșterea funcției de adaptare a liderului (adică îmbunătățirea succesivă a soluției optime).

În program, toate acestea se modelează astfel:

```
% curatenie
% initializari
% generare populatie
% vector numeric cuvant cautat (vector de referinta)

% vector functii de adaptare fgbest pentru reprezentare grafica
repr_fgbest=[];

for iter=1:nrit
    % calcul functie de adaptare
    % calcul pbest si gbest
    for i=1:np % pentru fiecare individ din populatie
        % recalculeaza viteza particulei
        % recalculeaza pozitie particula si rotunjeste valorile
        % valideaza particula
        % inlocuieste particula veche din populatie cu particula noua
        % si viteza veche cu viteza noua
    end;

    % afisari la sfarsitul fiecarei iteratii
    disp ('Iteratia:');
    disp(iter);
    disp('Solutia optima descoperita:');
    disp(gbest); % codurile ASCII
    disp('Cuvant ghicit:');
    disp(char(gbest)) % cuvantul
    disp('                    ')
    disp('Functia de adaptare a solutiei optime:');
    disp(fgbest);
    disp('                    ')

    % reprezentare grafica
    % adauga valoarea curenta a fgbest la vectorul pentru reprezentarea
    % grafica
    repr_fgbest=[repr_fgbest fgbest]; % adauga valoare in vector
    plot(repr_fgbest); % realizare grafic
    xlabel('iteratia'); % eticheta axa x
    ylabel('diferenta intre cuv. de ghicit si cuv. gasit'); % eticheta axa
y
    title ('Functia de adaptare PSO ghicire cuvant'); % titlu grafic
    drawnow;
    pause; % asteapta apasarea unai taste pentru a continua
end;
```

Comanda `drawnow` (desenează acum) forțează redesenarea imediată a graficului. Dacă nu se folosește această comandă, programul va face o reprezentare grafică la început, după prima iterație, când se inițializează figura, și una tocmai la final, după terminarea iterațiilor, fără reîmprospătări intermediare.

Comanda `pause` așteaptă apăsarea unei taste, pentru a trece la iterația următoare. Dacă nu se dorește vizualizarea optimului pas cu pas, ea poate fi comentată, iar în acest caz algoritmul va rula foarte repede până la sfârșitul iterațiilor.

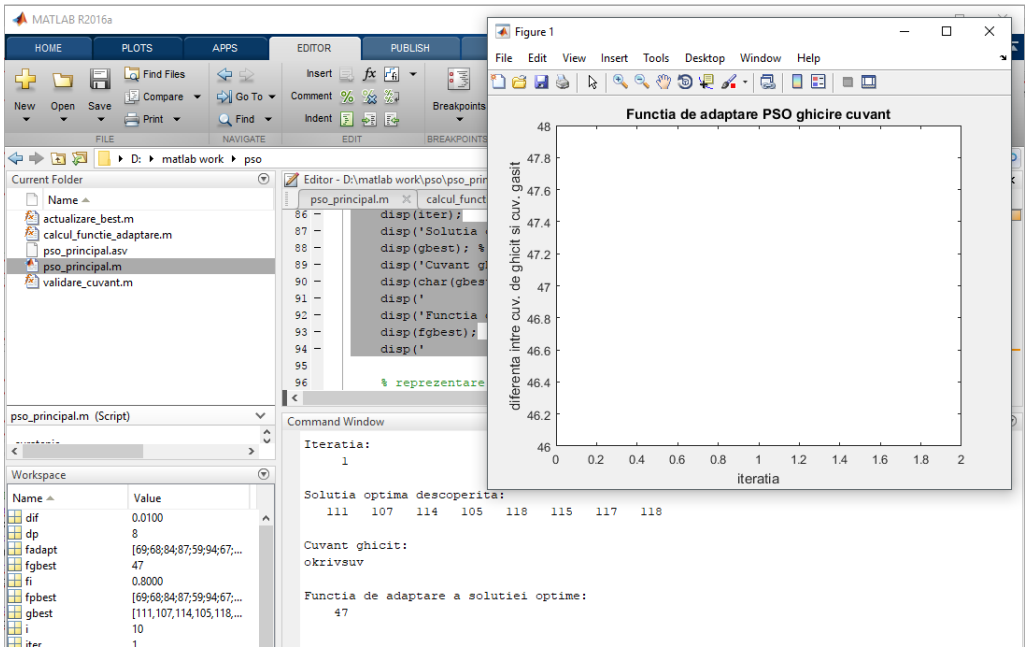
Algoritmul final arată astfel. El folosește cele trei funcții create mai devreme și populația inițială pentru teste.

```
% curatenie
clear;
clc;

% nr particule in populatie
np=10;

% numarul de iteratii
nrit=50;
 % dimensiunea cuvantului cautat (a unei particule)
dp=8;

% populatia initiala pbest
pbest=rand(np,dp);
% functiile de adaptare ale populatiei pbest
fpbest=10e6*ones(np,1);

% liderul initial al roiului
gbest=zeros(1,dp);
% functia de adaptare initiala a liderului
fgbest=10e6;

% vitezele initiale
vit=0.01*rand(np,dp);

% limite valori elemente individ
linf=97;
lsup=122;

% 2. Generare populatie
% generare populatie
%pop=round(linf+rand(np,dp)*(lsup-linf));

% populatia initiala pentru teste
pop=[99,121,98,100,98,102,98,113;
    104,111,119,113,116,103,114,110;
    117,110,120,109,110,119,98,121;
    98,103,117,116,109,98,99,113;
    120,109,99,115,120,109,110,117;
    115,113,104,120,112,101,99,108;
    109,114,105,119,112,121,117,108;
    111,107,114,105,118,115,117,118;
    103,106,100,114,117,110,115,99;
    108,122,115,102,111,109,101,100];

% vector numeric cuvant cautat (vector de referinta)
vref=double('albastru');


% factor initial de inertie
fi=0.9;
% rata de actualizare a factorului de inertie
dif=(0.9-0.4)/nrit;

% vector functii de adaptare fgbest pentru reprezentare grafica
```

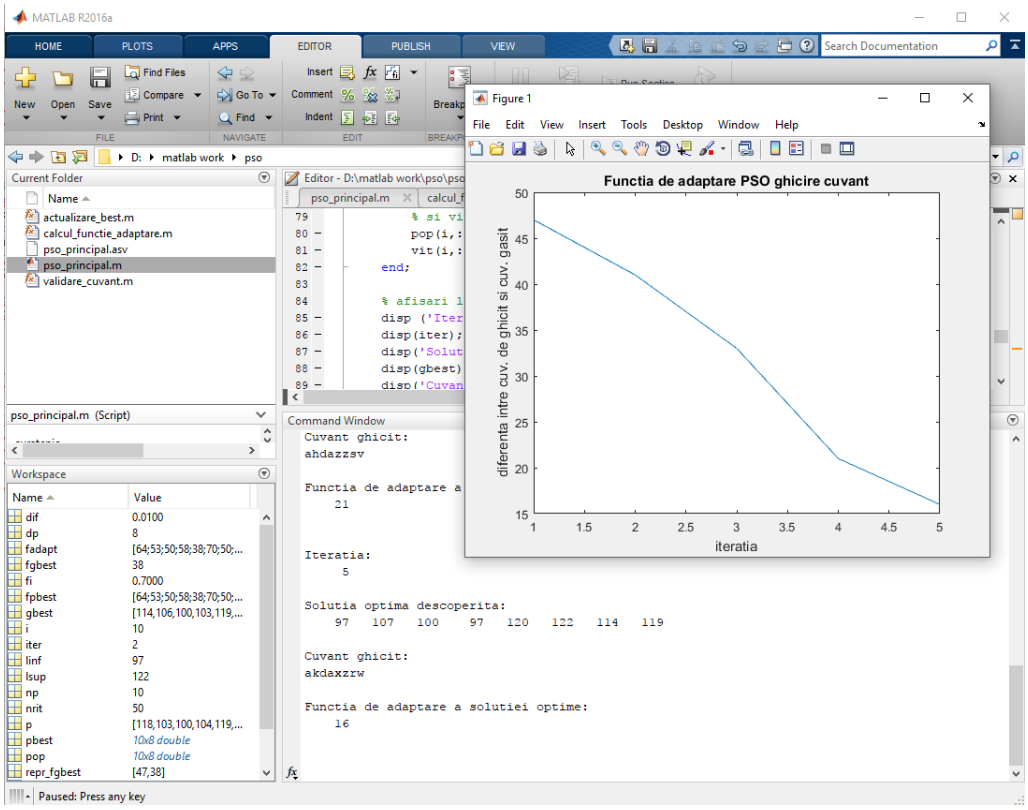
```
repr_fgbest=[];

for iter=1:nrit
    % calcul functie de adaptare
    [fadapt]=calcul_functie_adaptare(np,pop,vref);

    % calcul pbest si gbest

[pbest,fpbest,gbest,fgbest]=actualizare_best(pop,fadapt,pbest,fpbest,gbest,fgbest);

    for i=1:np % pentru fiecare individ din populatie
        % recalculeaza factor inertie
        fi=fi-dif;
        % recalculeaza viteza particulei
        v=fi*vit(i,:)+2*rand(1,dp).*(pbest(i,:)-pop(i,:))+2*rand(1,dp).*(gbest-
pop(i,:));

        % recalculeaza pozitie particula si rotunjeste valorile
        p=round(pop(i,:)+v);

        % validare particula
        [p]=validare_cuvant(dp,p,linf,lsup);

        % inlocuieste particula veche din populatie cu particula noua
        % si viteza veche cu viteza noua
        pop(i,:)=p;
        vit(i,:)=v;
    end;

    % afisari la sfarsitul fiecarei iteratii
    disp ('Iteratia:');
    disp(iter);
    disp('Solutia optima descoperita:');
    disp(gbest); % codurile ASCII
    disp('Cuvant ghicit:');
    disp(char(gbest)) % cuvantul
    disp('                      ')
    disp('Functia de adaptare a solutiei optime:');
    disp(fgbest);
    disp('                      ')
')

    % reprezentare grafica
    % adauga valoarea curenta a fgbest la vectorul pentru reprezentarea
    % grafica
    repr_fgbest=[repr_fgbest fgbest]; % adauga valoare in vector
    plot(repr_fgbest); % realizare grafic
    xlabel('iteratia'); % eticheta axa x
    ylabel('diferenta intre cuv. de ghicit si cuv. gasit'); % eticheta axa y
    title ('Functia de adaptare PSO ghicire cuvant'); % titlu grafic
    drawnow;
    pause; % asteapta apasarea unai taste pentru a continua
end;
```

Notă: atunci când veți replica exemplul, veți obține alte rezultate, din cauza vitezelor inițiale aleatorii și a componentelor aleatorii r1 și r2 din formula calculului vitezei, care probabil vor avea alte valori.

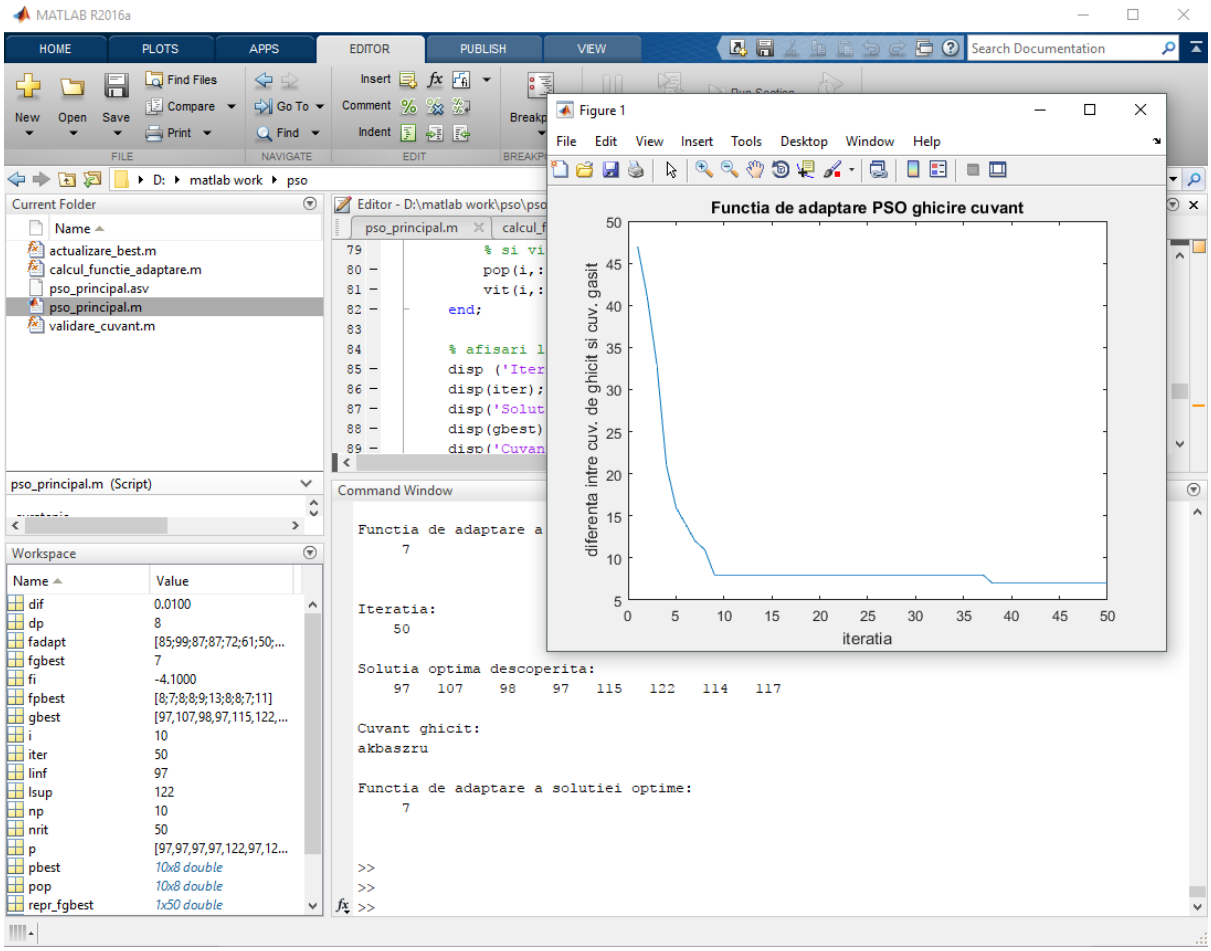
Rulând acest algoritm, după prima iterație, valoarea funcției de adaptare a liderului este cea așteptată, 47. Reprezentarea grafică conține doar un punct, deocamdată invizibil.



După 5 iterații, cuvântul ghicit este nepronunțabil, dat funcția de adaptare a scăzut la valoarea 16.



La sfârșitul celor 50 de iterații valoarea funcției de adaptare a cuvântului găsit este 7. Nu este soluția optimă, dar cuvântul este aproape de cel căutat.



Rulând algoritmul de zece ori cu 50 de iterații și de zece ori cu 100 de iterații, am obținut următoarele soluții optime, cu funcțiile lor de adaptare:

Nr rulare	Cuvânt găsit, rulare 50 de iteratii	Valoare funcție de adaptare, rulare 50 de iterații	Cuvânt găsit, rulare 100 de iteratii	Valoare funcție de adaptare, rulare 100 de iterații
1.	alcaptqu	5	alaastzu	9
2.	alaaqtru	3	akbaptru	4
3.	albassru	1	alaaszru	7
4.	alaatvru	4	alaastzu	9
5.	ajaastrw	5	alaaszru	7
6.	alaazzru	14	alaartrz	7
7.	alaastuu	4	alaastrz	6
8.	amaastru	2	alaastru	1
9.	alabluot	14	alaastru	1
10.	aocasttt	7	alaastrz	6

Ne-am apropiat doar de patru ori de soluția optimă, dar n-am atins-o niciodată. Creșterea numărului de iterații a îmbunătățit rezultatul, dar nu suficient. Algoritmul PSO se comportă conform teoriei metodelor metaeuristice: găsește soluții apropiate de optimul global, fără să-l atingă neapărat.

Una dintre cauzele acestei comportări poate fi dimensiunea populației. PSO funcționează optim cu 30-50 de particule în populație, iar noi am folosit doar zece.

O altă cauză ar putea fi felul în care se face validarea. Se observă destul de multe litere „z” și „a” în soluțiile optime, care ar putea fi obținute prin limitarea la pragul superior admis al unor valori mai mari de 122, respectiv mai mici decât 97 din particule.

Recapitulare

Am adus algoritmul PSO la o formă funcțională. Am completat procesul iterativ și am adăugat afișări de rezultate. Deocamdată, algoritmul nu reușește să descopere soluția optimă globală, cuvântul „albastru”.

Va urma

Vom urmări în detaliu câteva iterații ale algoritmului, pentru a monitoriza evoluția populației și a depista eventuale greșeli ascunse. Vom crește numărul de indivizi din populație și vom schimba metoda de validare a particulelor, pentru a vedea dacă acestea influențează precizia rezultatelor obținute.