Algoritmi metaeuristici.

Lab 1: Probleme de optimizare

Familiarizare cu Scilab

---

1. Probleme de optimizare

Problemele de optimizare reprezintă una dintre clasele de probleme cel mai frecvent întâlnite în aplicații care necesită:

- *Planificare* (planificarea și/sau stabilirea ordinii de execuție a unor activități, alocarea de resurse unor activități)
- *Modelare* (construirea unui model care să se potrivească cât mai bine cu datele experimentale)
- *Adaptare* (modificarea comportamentului unui sistem astfel încât să aibă comportamentul dorit)

Problemele de optimizare sunt caracterizate prin: spațiu de căutare a soluțiilor, criteriu de optim (funcția obiectiv), restricții.

1.1. ***Spațiul de căutare a soluțiilor.*** Principalele variante sunt:

- spațiu discret de căutare (de regulă o mulțime finită dar de dimensiuni mari).
- spațiu continuu de căutare

Din punctul de vedere al spațiului de căutare problemele de optimizare se grupează în:

- Probleme de optimizare combinatorială
  - Problema de alocare/selecție (exemplu: problema rucsacului, problema împachetării)
  - Probleme de ordonanțare/rutare (exemplu: problema comis voiajorului)
- Probleme de optimizare continua
  - Estimarea parametrilor unor modele (exemplu: determinarea reprezentanților în probleme de grupare a datelor, antrenarea rețelelor neuronale)

1.2. ***Criteriul de optim (funcția obiectiv).*** Reprezintă valoarea care trebuie optimizată (minimizată sau maximizată). In funcție de numărul de criterii de optimizat problemele pot fi de:

- Optimizare unicriterială (o singură funcție obiectiv)
- Optimizare multicriterială (mai multe funcții obiectiv)

In funcție de informațiile disponibile despre funcția obiectiv, problemele de optimizare pot fi de tip:

- *White-box:* funcția obiectiv este cunoscută explicit și pot fi analizate și exploatate proprietățile ei (liniaritate/neliniaritate, continuitate/discontinuitate, netezime etc).
- *Black-box:* funcția obiectiv nu este cunoscută explicit ci doar se pot calcula valorile ei pentru elemente din spațiul de căutare.

1.3 ***Restricții.*** Restricțiile definesc subspațiul soluțiilor fezabile si pot fi de diferite tipuri:

- Restricții de mărginire (domeniu): a<=x<=b

- Restricții de tip egalitate: g(x)=a
- Restricții de tip inegalitate: h(x)<=b

In funcție de importanța restricțiilor acestea pot fi:

- Stricte: trebuie neapărat să fie satisfăcute
- Relaxate: e preferabil să fie satisfăcute însă încălcarea lor nu conduce la soluții nefezabile ci doar la soluții de calitate mai scăzută

2. Etape în rezolvarea problemelor de optimizare

Etapa 1: analiza problemei și identificarea următoarelor elemente:

- Structura (modul de reprezentare) a soluțiilor candidat; aceasta permite stabilirea proprietăților și dimensiunii spațiului de căutare. Problema poate fi încadrată în categoria:
  - Optimizare combinatorială
  - Optimizare continuă

  de dimensiune
  - mică (sub 10 variabile)
  - medie (între 10 și 100 de variabile)
  - mare (peste 100 de variabile)

- Restricții
- Funcția obiectiv și proprietățile acesteia:
  - Lineară, pătratică, arbitrară
  - Continuă/discontinuă, diferențiabilă/nediferențiabilă
  - O singură funcție/ mai multe funcții, funcție unimodală/multimodală

Etapa 2: selectarea metodei:

- Metode specifice cu funcții obiectiv/restricții liniare/pătratice
- Optimizare locală care utilizează derivate
- Optimizare locală care nu utilizează derivate
- Optimizare globală
- Optimizare multicriterială

3. Optimizare in SciLab

Objective	Bounds	Equality	Inequalities	Size	Gradient Needed	Solver
Linear	yes	linear	linear	medium	-	linpro
Quadradic	yes	linear	linear	medium	-	quapro
Quadratic	yes	linear	linear	large	-	qpsolve
Quadratic	yes	linear	linear	medium	-	qld
Non-Linear	yes			large	yes	optim
Non-Linear				small	no	fminsearch
Non-Linear	yes			small	no	neldermead
Non-Linear	yes			small	no	optim_ga
Non-Linear				small	no	optim_sa
N.Li.Lea.Sq.				large	optional	lsqrsolve
N.Li.Lea.Sq.				large	optional	leastsq
Min-Max	yes			medium	yes	optim/nd
Multi-Obj	yes			small	no	optim_moga
Semi-Def.			lin. (spectral)	large	no	semidef
L.M.I.		lin. (spectral)	lin. (spectral)	large	no	lmisolve

Functii Scilab pentru diferite clase de probleme de optimizare [M. Baudin, V. Couvert, Optimization with SciLab, 2011]

3.1. Optimizare folosind funcția optim

Tipuri de probleme care pot fi rezolvate: probleme de optimizare neliniară fără restricții (sau doar cu restricții de mărginire a domeniului).

Metode pe care se bazează: metode de tip cvasi-Newton (sunt metode de optimizare locală care necesită specificarea unei aproximații initiale și a derivatelor în raport cu fiecare dintre variabile)

Moduri de apel:

```
[xopt,fopt]=optim(fctieObiectiv, x0)      // x0 = aproximatie initiala
[xopt,fopt]=optim(fctieObiectiv,"b",liminf,limsup, x0)   // liminf, limsup = vectori cu limitele
                                                          //domeniului
[xopt,fopt]=optim(fctieObiectiv,x0, alg)   //alg=algoritm
        //"qn": cvasi-Newton (implicit) bazat pe BFGS (Broyden-Fletcher-Goldfarb-Shanno)
        //"gc": BFGS cu memorie limitată (adecvat pentru număr mare de variabile)
        //"nd": pentru funcții obiectiv nediferențiabile
```

Obs: funcția SciLab care implementează criteriul de optim trebuie să returneze atât valoarea funcției obiectiv cât și gradientul acesteia.

Exemplu: Să se estimeze punctul de optim al funcției, f:RxR->R, aflat în vecinătatea lui (-1,1.5).

$f(x1,x2)=100*(x2-x1^2)^2 +(1-x1)^2$

(funcția e cunoscută sub numele de Rosenbrock şi este considerată o funcție dificil de optimizat în cazul unui număr mare de variabile)

Implementare SciLab:

```
function [f, g, ind]=rosenbrock(x, ind)
  f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2          // fctie obiectiv
  g(1) = - 400*(x(2)-x(1)^2)*x(1) - 2*(1-x(1)) // componente ale
                                                //gradientului
  g(2) = 200*(x(2)-x(1)^2)
endfunction
x0 = [-1.0 1.5];    // aproximatia initiala
[ fopt , xopt ] = optim ( rosenbrock , x0)
```

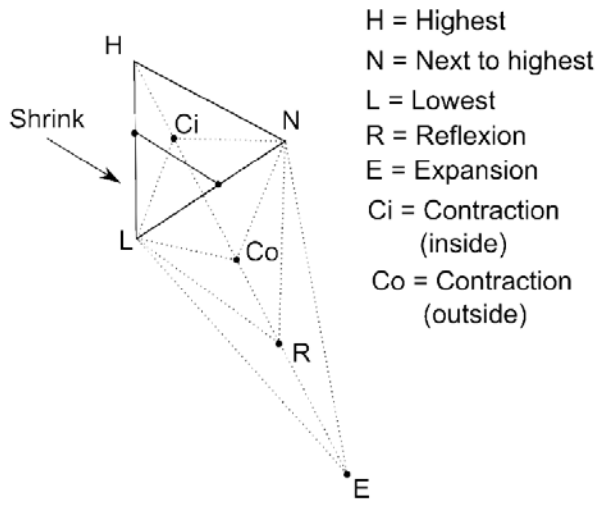
Exercițiu 1.

a) Să se compare rezultatele obținute utilizând “qn”, “gc”, “nd”
b) Să se analizeze cazul în care derivatele nu se specific analitic ci se estimează numeric:
   g=numderivative(rosebrockF,x)
   In apelul funcției numderivative, rosenbrockF e o functie care returnează doar valoarea corespunzătoare unui argument (nu și gradientul)
c) Să se modifice pentru cazul unui număr arbitrar de variabile

3.2. Optimizare folosind fminsearch

Tipuri de probleme care pot fi rezolvate: probleme de optimizare neliniară fără restricții pentru funcții nediferențiabile.

Metode pe care se bazează: metoda Nelder-Mead (căutarea se bazează pe construirea de noi soluții candidat utilizând o structură de tip simplex și o serie de transformări – vezi fig de mai jos si curs 2)



Moduri de apel:

```
[xopt,fopt]=fminsearch(fctieObiectiv, x0)      // x0 = aproximatie initiala
[xopt,fopt]=fminsearch(fctieObiectiv, x0, optiuni)
```

Optiunile se seteaza cu funcția optimset: optiuni= optimset(numeOptiune,valoareOptiune)
Tipuri de optiuni:

- MaxIter - numar maxim de iterații (implicit: 200*nr variabile)
- MaxFunEvals – număr maxim de evaluări ale funcției obiectiv (implicit: 200*nr variabile)
- TolFun – toleranța asociată valorilor funcției obiectiv (implicit: 0.0001)
- TolX – toleranța asociată variabilelor (implicit: 0.0001)
- PlotFcns – vizualizare grafică (implicit: empty – fără vizualizare)

Exemplu: optimizarea funcției Rosenbrock

Implementare SciLab:

```
function f=rosenbrockF(x)
  f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2          // fctie obiectiv

endfunction
x0 = [-1.0 1.5];    // aproximatia initiala
[ fopt , xopt ] = fminsearch ( rosenbrockF , x0)
```

Exercițiu 2. Implementați și testați următoarele variante:

a) Vizualizarea grafică a procesului de minimizare:

```
optiuni = optimset ( "PlotFcns" , optimplotfval )
x0=[-1,1.5]
[ fopt , xopt ] = fminsearch ( rosenbrockF , x0, optiuni)
```

b) Vizualizarea transformărilor aplicate la fiecare iterație:

```
optiuni = optimset ( "Display" , iter )
x0=[-1,1.5]
[ fopt , xopt ] = fminsearch ( rosenbrockF , x0, optiuni)
```

c) Testarea cazului în care sunt mai multe variabile

Anexa: Scilab - Open source software for numerical computation (http://www.scilab.org/)

Scilab este un limbaj de programare interpretat care oferă suport pentru prelucrări specifice din algebra liniară, pentru prelucrări asupra funcțiilor polinomiale și a celor raționale, interpolare și aproximare, optimizare liniară, optimizare pătratică, rezolvarea ecuațiilor diferențiale, prelucrarea semnalelor, statistică și prelucrări grafice. Ca principiu de lucru dar și ca interfață (în ultimele versiuni) este similar pachetului Matlab.

In Scilab, la fel ca în Matlab, obiectul principal este *matricea*, atât vectorii cât și scalarii fiind considerați cazuri particulare de matrici.

Aspecte generale:

- Scilab este case-sensitive
- Fiind interpretor, variabilele nu trebuie declarate însă trebuie să aibă asignată o valoare; operatorul de asignare se specifică prin =
- Constantele predefinite au numele prefixat de % (de exemplu: %pi, %i, %e, %t (true), %f (false))
- Operatorii relaționali sunt: == (egal), ~= sau <> (diferit), <=, >=
- Operatorii logici sunt: ~ (not), & (and), | (or)
- Rezultatul unei evaluări este asignat implicit variabilei ans care poate fi utilizată în comanda imediat următoare
- Comenzile plasate pe aceeași linie trebuie separate prin ; (separatorul ; are și efectul inhibării vizualizării rezultatului ultimei evaluări)
- Șirurile de caractere se încadrează între ghilimele (") iar operatorul de concatenare este +
- Comentariile de tip linie se specifică prin //

Specificarea/definirea matricilor.

- Explicit prin enumerarea elementelor (elementele de pe aceeași linie se separă prin virgulă (sau spațiu) iar liniile se separă prin punct-virgula (sau enter):
  A=[a11,a12,…a1n; a21,a22,…,a2n; …;am1, am2,…,amn]

- Implicit prin utilizarea unor funcții care generează matrici:
  - zeros(m,n) : matrice cu m linii și n coloane și elemente egale cu 0
  - ones(m,n): matrice cu m linii și n coloane și elemente egale cu 1
  - rand(m,n): matrice cu m linii și n coloane și elemente generate uniform aleator în intervalul (0,1)

Operații asupra matricilor.

- *Determinarea dimensiunii:* size(matrice) returnează vectorul [nr linii, nr coloane]
- *Reorganizarea unei matrici:* matrix(matrice,nrlinii,nrcoloane) returnează o matrice cu numărul de linii și de coloane specificat și elementele din matricea inițială
- *Specificarea elementelor:* matrice(indice linie,indice coloana).
  Obs: indicii pot fi atât valori individuale cât și domenii de valori specificate prin inf:sup. Pentru a specifica ultimul indice de linie/coloană se poate utiliza $. De exemplu matrice($,$-1) indică elementul de pe ultima linie, penultima coloană.
  Obs: indicii elementelor încep cu 1
- *Modificarea unei matrici:*
  - Modificare element: matrice(i,j)=valoare
  - Adăugare linie: matrice=[matrice; el1, el2,…,eln]

- Adăugare coloana: matrice=[matrice'; el1, el2,…,elm]' (operatorul ' corespunde calculului transpusei matricii)
- Ștergere linie: matrice(i,:)=[]
- Ștergere coloană: matrice(:,j)=[]

- *Operații aritmetice:* toate operațiile aritmetice sunt vectorizate; pentru a efectua operații la nivel de element trebuie specificat . (punct) înainte de operator. De exemplu A*B returnează produsul matricilor A și B iar A.*B returnează matricea în care pe linia i coloana j se află produsul elementelor aflate pe aceeași poziție în cele două matrici. Este indicat să fie folosite operații vectorizate în locul iterării explicite a unor operații scalare întrucât sunt mai eficiente (de la 10 până la 100 ori mai rapide).

Alte tipuri de obiecte în Scilab:

- *Structuri:*
  - struct(numeCâmp1,valoare1,numeCâmp2,valoare2,….,numeCâmpn, valoaren)
  - Exemplu: data=struct('zi',30,'luna','septembrie','an',2014)
  - Specificare elemente: numeStructura.numeCâmp (de exemplu: data.zi este 30)

- *Liste heterogene:*
  - *Listă simplă*: list(Element1,Element2,…,Elementn)
    Obs: elementele se specifică utilizând indici
  - *Listă cu tip*: tlist(listaNumeElemente,Element1,Element2,…,Elementn) ;
    Exemplu: d=tlist(['data','zi','luna','an'],20,'sept',2014)
    Obs: elementele pot fi specificate atât prin indiciere cât și prin calificare: d(2) este identic cu d.zi

Instrucțiuni de control (ramificare și ciclare)

- **Instrucțiunea if:**

```
if (conditie) then
        <Prelucrare 1>
else
        <Prelucrare 2>
end
```

Varianta imbricată:

```
if (conditie1) then
        <Prelucrare 1>
elseif (conditie 2)
        <Prelucrare 2>
else
        <Prelucrare 3>
end
```

- **Instrucțiunea select:**

```
select <variabila selector>
case <valoare 1>
        <prelucrare 1>
```

```
case <valoare 2>
        <prelucrare 2>
…
case <valoare n>
        <prelucrare n>
else
        <alta prelucrare>
end
```

- **Instrucțiunea for**

```
for contor=inf:pas:sup
        <Prelucrare>
end
```

Obs: Dacă pasul este 1 atunci poate fi omis din specificarea domeniului. Valoarea pasului poate fi și număr negativ. Iterarea se poate face și pe elementele unui vector (matrice linie):

```
for contor=vector
        <Prelucrare>
end
```

- **Instrucțiunea while**

```
while(conditie)
        <Prelucrare>
end
```

Definirea și apelul funcțiilor

Funcțiile pot fi utilizate pentru a calcula mai multe rezultate (specificate prin variabilele de ieșire indicate la definirea funcției):

```
function [output1, …,outputm]=numeFunctie(input1,…,inputn)
        <corp functie>
endfunction
```

Prelucrări grafice

Principalele funcții grafice din Scilab sunt:

- plot - pentru reprezentarea grafică a funcțiilor uni-dimensionale
- fplot3d și contour - pentru reprezentarea grafică a suprafețelor
- paramfplot2d – reprezentare curbe descries prin ecuații parametrice
- polarplot – reprezentare în coordonate polare

```
Exemple:
// graficul unei functii
function y=f(x)
    y=x*x/10+sin(x)
endfunction

x=-2*%pi:0.1:2*%pi
```

```
clf
plot(x,f)

// graficul unei suprafete
function y=f2arg(x1, x2) // functie cu doua variabile
    y = x1 **2 + x2 **2;
endfunction
x1data = linspace ( -1 , 1 , 100 );
                        // echivalent cu x1data = -1:0.1:1;
x2data = linspace ( -1 , 1 , 100 );
contour ( x1data , x2data , f2arg , 10)  // contour plot
pause
clf                                      // curatire ecran
fplot3d( x1data , x2data , f2arg)        // surface plot
```