CURS 8: METODE DE OPTIMIZARE PARAMETRICĂ

Problemele de optimizare vizează extremizarea (maximizarea sau minimizarea) unui ***criteriu de performanţă***. Acesta din urmă poate fi o *funcţie* – caz în care este vorba despre *optimizarea parametrică* – sau o *funcţională* (o funcţie de funcţii), când este vorba despre *optimizarea dinamică*. Acest paragraf este dedicat ilustrării comparative a principalilor algoritmi de optimizare parametrică.

1.1 Formularea problemei. Definiţii

Criteriul de performanţă care trebuie extremizat se mai numeşte ***funcţie scop***, ***funcţie obiectiv*** sau ***funcţie criteriu*** fără impuneri de legături sau/şi restricţii. Aceasta este o funcţie de n argumente reale, $f(x_1, x_2, ...x_n)$, a cărei extremizare înseamnă găsirea unui vector $\underline{x}^* = \begin{bmatrix} x_1^* & x_2^* & ... & x_n^* \end{bmatrix}^T \in \mathbb{R}^n$, corespunzător unui punct din spaţiul n-dimensional, pentru care funcţia este *optimă* (maximă sau minimă). Maximizarea şi minimizarea unei funcţii nu se deosebesc formal, întrucât maximul lui f se obţine pentru minimul lui $-f$.

Pentru $n \leq 2$, funcţia obiectiv admite o reprezentare geometrică, şi anume:

- bidimensională: $y = f(x_1)$ este ecuaţia unei curbe în planul (x_1, y);
- tridimensională: $y = f(x_1, x_2)$ este ecuaţia unei suprafeţe în sistemul de coordonate (x_1, x_2, y) (*suprafaţa de răspuns*).

Generalizarea acestor reprezentări pentru o funcţie obiectiv de n variabile independente necesită un sistem de $n+1$ axe independente, perpendiculare două câte două într-un sistem cartezian. Pentru $n \geq 3$ aceasta este evident imposibil de realizat în spaţiul tridimensional şi de aceea dezvoltările teoretice – cu introducerea noţiunilor de „hiperplan", „hipersuprafaţă", „hiperspaţiu" – sunt uneori mai dificil de urmărit.

Este utilă reluarea a două definiţii cunoscute din matematică.

Definiţia 1: Fie funcţia scalară *continuă* şi *derivabilă* de variabilă vectorială $f(\underline{x})$, $\underline{x} \in \mathbb{R}^n$. Se numeşte ***gradientul*** funcţiei f funcţia *vectorială* de variabilă vectorială notată cu $\nabla f(\underline{x})$, care conţine derivatele parţiale de ordinul întâi ale lui f în raport cu fiecare dintre componentele lui $\underline{x}$:

$$(1) \qquad f'(\underline{x}) = \nabla f(\underline{x}) \triangleq \begin{bmatrix} \frac{\partial f(\underline{x})}{\partial x_1} & \frac{\partial f(\underline{x})}{\partial x_2} & ... & \frac{\partial f(\underline{x})}{\partial x_n} \end{bmatrix}^T$$

Funcţia $-\nabla f(\underline{x})$ se numeşte ***antigradientul*** funcţiei f. Punctele de *extrem*, $\underline{x}^*$, ale unei funcţii de variabilă vectorială sunt caracterizate de *anularea gradientului*: $\nabla f(\underline{x}^*) = 0$. Direcţia vectorului gradient indică spre maximul unei funcţii concave, iar a antigradientului indică spre minimul unei funcţii convexe.

Definiţia 2: Se numeşte ***(matricea) hessian*** a funcţiei de variabilă vectorială $f(\underline{x})$ funcţia *matricială* de variabilă vectorială notată cu $H(\underline{x})$, obţinută printr-o nouă derivare a gradientului în raport cu componentele vectorului $\underline{x}$:

$$(2) \qquad f''(\underline{x}) = H(\underline{x}) \triangleq \begin{bmatrix} \frac{\partial^2 f(\underline{x})}{\partial x_1^2} & \frac{\partial^2 f(\underline{x})}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(\underline{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\underline{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\underline{x})}{\partial x_2^2} & \dots & \frac{\partial^2 f(\underline{x})}{\partial x_2 \partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 f(\underline{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\underline{x})}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(\underline{x})}{\partial x_n^2} \end{bmatrix}$$

1.2 Clasificarea metodelor de optimizare parametrică

Principalele metode de optimizare parametrică sunt sintetizate în tabelul 2.9; ele sunt detaliate în paragrafele următoare. Cele două mari clase de metode – *indirecte* şi *directe* – se deosebesc atât principial, cât şi din punctul de vedere al performanţelor (viteza de convergenţă, timpul de calcul şi memoria necesară algoritmilor rezultaţi). Astfel, metodele directe sunt în general mai *lent convergente* decât metodele indirecte, dar necesită un volum de memorie mai mic. Nivelul admisibil al compromisului acceptat în ce priveşte performanţele este un factor în alegerea uneia sau alteia dintre metode.

<table>
<tr><td rowspan="2">A. indirecte
(folosesc valorile funcţiei $f(\underline{x})$ şi pe cele ale derivatelor ei)</td><td colspan="2">A.1 metode de gradient (Cauchy)</td></tr>
<tr><td colspan="2">A.2 metode Newton (de gradient de ordinul al II-lea)</td></tr>
<tr><td></td><td colspan="2">A.3 metode ale direcţiilor conjugate</td></tr>
<tr><td rowspan="5">B. directe
(folosesc numai valorile funcţiei $f(\underline{x})$, fără a le utiliza pe cele ale derivatelor ei)</td><td colspan="2">B.1 metoda explorării exhaustive</td></tr>
<tr><td rowspan="2">B.2 metode de eliminare</td><td>unidimensională</td></tr>
<tr><td>multidimensională</td></tr>
<tr><td colspan="2">B.3 metode de căutare pe bază de hiperpoliedre (simplexuri) exploratoare</td></tr>
<tr><td colspan="2">B.4 metode de căutare aleatoare (Monte Carlo)</td></tr>
<tr><td colspan="2">B.5 metode de căutare unidirecţională (unidimensională) ≡ metode de relaxare (Gauss)</td></tr>
</table>

Tabel 1 Clasificarea metodelor de optimizare parametrică

1.3 Metode indirecte

Metodele indirecte se mai numesc şi *metode de urcare (coborâre)*; pentru simplificare, pintr-un lejer abuz de limbaj, termenul de „metode de gradient" denotă în mod generic întreaga clasă a metodelor indirecte. Esenţa lor constă în ***găsirea punctului de anulare a gradientului*** (acesta fiind punctul de extrem căutat), pornind dintr-un punct iniţial dat. În calculul aproximativ, termenul de „anulare" înseamnă, de fapt, situarea valorii absolute sub o anume limită (toleranţă) dată, considerată ca suficientă.

Metodele de gradient (Cauchy) şi cele de gradient de ordinul al II-lea (Newton) se

bazează pe aproximări de ordinul I, respectiv de ordinul al II-lea, ale dezvoltării în serie Taylor a funcţiei obiectiv $f(\underline{x})$ în jurul punctului de optim, $\underline{x}^*$:

$$(3) \qquad f(\underline{x}) \cong f\left(\underline{x}^*\right) + f'\left(\underline{x}^*\right)^T \cdot \left(\underline{x} - \underline{x}^*\right)$$

$$(4) \qquad f(\underline{x}) \cong f\left(\underline{x}^*\right) + f'\left(\underline{x}^*\right)^T \cdot \left(\underline{x} - \underline{x}^*\right) + \frac{1}{2} \cdot \left(\underline{x} - \underline{x}^*\right)^T \cdot f''\left(\underline{x}^*\right)^T \cdot \left(\underline{x} - \underline{x}^*\right)$$

Metodele Newton au o convergenţă mai bună decât cele de gradient simplu, dar prezintă inconvenientul unui timp de calcul şi al unui volum de memorie mai mari, pentru că necesită calculul matricei hessian la fiecare iteraţie de căutare.

În formula (23) gradientul arată direcţia ratei maxime de creştere a funcţiei *f*. Pe această formulă se bazează ***metoda celei mai mari pante*** (dacă optimul este un ***maxim***) sau ***metoda celei mai abrupte coborâri*** (dacă optimul este un ***minim***). Mai jos se prezintă descrierea în meta-limbaj a celui mai simplu algoritm de gradient (notaţia ||·|| semnifică norma vectorială).

Date de intrare: ε (toleranţa suficientă), $\underline{x}^{(0)}$ (punctul iniţial)
$i \leftarrow 0$
Repetă
#1. Se determină direcţia de căutare la pasul *i* (dată de versorul asociat gradientului):

$$(5) \qquad \underline{d}^{(i)} = \pm \frac{\nabla f\left(\underline{x}^{(i)}\right)}{\left\|\nabla f\left(\underline{x}^{(i)}\right)\right\|}, \text{ unde „+/–” corespund maximizării/minimizării}$$

#3. Se alege arbitrar $p^{(i)}$, pasul de deplasare pe direcţia $\underline{d}^{(i)}$.
#2. Se calculează noul punct de evaluare a gradientului:

$$(6) \qquad \underline{x}^{(i+1)} = \underline{x}^{(i)} + p^{(i)} \cdot \underline{d}^{(i)}$$

$i \leftarrow i+1$
Până când $\left\|\nabla f\left(\underline{x}^{(i)}\right)\right\| \le \varepsilon$.

Se observă că algoritmul de mai sus necesită cunoaşterea a priori a tipului de extrem căutat (maxim sau minim). Implementarea în Matlab se poate face sub forma unei funcţii care primeşte punctul de start al căutării, `x0` (la cazul general acesta este un vector) şi toleranţa dorită, `epsilon`, şi returnează valoarea de extrem a unei funcţii (la cazul general, vectoriale) cunoscute, `x`, şi numărul de iteraţii în care s-a obţinut aceasta, `nr_it`. S-a utilizat o variabilă locală, `itermax`, pentru a forţa ieşirea din ciclare dacă extremul nu poate fi găsit.

```
function [x,nr_it]=opt_grad(x0,epsilon)
   nr_it=1;x_curent=x0;itermax=1500;%număr maxim de iteraţii
   h=0.001; %variaţie utilizată în calculul gradientului
   p=0.001;%se alege un pas constant de deplasare
   for k=1:n,
      v=zeros(1,n);v(k)=1;
      grad_curent(k)=(f1(x0+h/2*v)-f1(x0-h/2*v))/h;
   end;
```

```
    while (norm(grad_curent)>=epsilon)&(nr_it<=itermax),
        %implementarea formulei (2.29) folosind (2.28);
        %pentru o funcţie de maximizat „–” se înlocuieşte cu „+”
        x_viitor=x_curent-p*grad_curent/norm(grad_curent);
        for k=1:n,
         v=zeros(1,n);v(k)=1;
         grad_viitor(k)=(f1(x_viitor+h/2*v)-f1(x_viitor-h/2*v))/h;
        end;
        x_curent=x_viitor;
        grad_curent=grad_viitor;
        nr_it=nr_it+1;
    end;
    x=x_curent;
```

Funcţia de mai sus face uz de apelul funcţiei Matlab `norm`, care calculează norma vectorială sau matricială; ea realizează *minimizarea* unei funcţii care trebuie să se afle în fişierul `f1.m`; în particular ea poate fi o funcţie scalară:

```
function y=f1(x)
    y=5*x^2+2*x+13;
```

Valoarea în care se atinge minimul acestei funcţii este –0.2, care se obţine după un număr de iteraţii mai mic sau mai mare, depinzând de condiţia iniţială şi de pasul de deplasare ales. Într-adevăr, apelul:

```
[x,nr_it]=opt_grad(1,1e-3)
```

produce rezultatele:

```
x =
   -0.2000

nr_it =
         1201
```

iar apelul:

```
[x,nr_it]=opt_grad(-1,1e-3)
```

are drept rezultat:

```
x =
   -0.2000

nr_it =
   801
```

Numărul mare de iteraţii se datorează folosirii unui pas foarte mic (p=0.001); folosirea unui astfel de pas se justifică în cazul extremelor abrupte („creste” ascuţite sau „văi” abrupte), unde componentele gradientului au variaţii mari.

Algoritmul Fletcher-Reeves (*metoda gradienţilor conjugaţi*) este o procedură puternică de determinare a *minimului* local al unei funcţii generale, $f(\underline{x})$. De această dată, la fiecare iteraţie i se defineşte o nouă direcţie de căutare, $\underline{p}^{(i)}$, ca o combinaţie liniară între vectorul gradient la iteraţia curentă, $\nabla f(\underline{x}^{(i)})$, şi direcţiile de la iteraţiile anterioare, $\left\{\underline{p}_j^{(i)}\right\}_{j=\overline{0,i-1}}$.

Etapele algoritmului sunt listate mai jos.

1. Se alege un punct iniţial caracterizat de vectorul $\underline{x}^{(0)}$.

2. Se stabileşte direcţia iniţială drept direcţia negativă a gradientului în $\underline{x}^{(0)}$:

$$\underline{p}^{(0)} = -\nabla\left(f\left(\underline{x}^{(0)}\right)\right)$$

3. La fiecare iteraţie *i* se determină minimul în raport cu parametrul α al funcţiei obiectiv în direcţia $\underline{p}^{(i)}$, conform relaţiei:

$$(7) \qquad f\left(\underline{x}^{(i+1)}\right) = f\left(\underline{x}^{(i)} + \alpha \cdot \underline{p}^{(i)}\right)$$

Se determină astfel punctul $\underline{x}^{(i+1)}$.

4. Se determină noua direcţie de căutare, $\underline{p}^{(i+1)}$, din punctul $\underline{x}^{(i+1)}$:

$$(8) \qquad \underline{p}^{(i+1)} = -\nabla f\left(\underline{x}^{(i+1)}\right) + \frac{\left\|\nabla f\left(\underline{x}^{(i+1)}\right)\right\|^2}{\left\|\nabla f\left(\underline{x}^{(i)}\right)\right\|^2} \cdot \underline{p}^{(i)}$$

5. *Dacă* $f\left(\underline{x}^{(i+1)}\right) \geq f\left(\underline{x}^{(i)}\right)$, *atunci* punctul de minim a fost găsit: $\underline{x}^{*} \leftarrow \underline{x}^{(i)}$, STOP;

altfel se reia de la etapa 3.

În cele ce urmează este dat programul Matlab (`fletreev.m`) care implementează algoritmul Fletcher-Reeves; acesta va fi testat pentru aceeaşi funcţie scalară de gradul al II-lea conţinută în fişierul `f1.m`, cu un minim la –0.2.

```
%iniţializări
i=1;x(i,:)=x0;n=length(x0);itermax=1000;%număr maxim de iteraţii
h=0.0001;%variaţie utilizată în calculul gradientului
for k=1:n,
   v=zeros(1,n);v(k)=1;
   %la fiecare pas gradientul este un vector de dimensiune n
   grad(i,k)=(f1(x0+h/2*v)-f1(x0-h/2*v))/h;
end;
p(i,:)=-grad(i,:);
stop=0;%variabilă booleană care arată găsirea minimului local căutat
while (~stop)&(i<=itermax),
   alfa=0:0.01:10;
   y_min=f1(x(i,:));
   %se determină minimul funcţiei când alfa variază în intervalul (ales arbitrar) [0;10]
   %cu pasul 0.01
   stop=1;
   for j=1:length(alfa),
      y=f1(x(i,:)+alfa(j)*p(i));%formula (2.30)
      if y<y_min,
         y_min=y;
         x(i+1,:)=x(i,:)+alfa(j)*p(i);
         stop=0;
      end;
```

```
      end;%în acest moment noul punct de căutare este x(i+1)
      if (~stop)
         for k=1:n,
            v=zeros(1,n);v(k)=1;
            grad(i+1,k)=(f1(x(i+1,:)+h/2*v)-f1(x(i+1,:)-h/2*v))/h;
         end;
         %noua direcţie de căutare se determină cu formula (2.31)
         p(i+1,:)=-grad(i+1,:)+...
                  norm(grad(i+1,:))^2/norm(grad(i,:))^2*p(i,:);
      end;
   i=i+1;
   end;

   [nr_it n]=size(x);nr_it
   minim=x(nr_it,:)
```

Execuţia programului din linie de comandă necesită iniţializarea variabilei `x0`, punctul de start al căutării. De exemplu, pentru:

```
   x0=1;fletreev
```

rezultatul:

```
   nr_it =
        2

   minim =
      -0.2000
```

arată o convergenţă mai rapidă decât cea a metodei de gradient anterioare, implementată prin funcţia utilizator `opt_grad`.

Observaţie:

Pentru o funcţie de *n* variabile care *nu* este pătratică, după fiecare *n* iteraţii se reiniţializează direcţia de căutare la direcţia antigradientului, ca la primul pas. Scopul este eliminarea erorilor datorate faptului că funcţia se poate aproxima bine cu o funcţie pătratică *numai* în stricta vecinătate a optimului.

1.4 Metode directe. Metode de relaxare

Din clasa metodelor directe de căutare a optimului, cea mai simplă, dar şi cea mai costisitoare ca timp, este cea a ***explorării exhaustive***. Metodele ***de eliminare*** se folosesc când funcţia obiectiv are *un singur optim (funcţie unimodală)*; ele se bazează pe eliminarea unei regiuni a domeniului de variaţie a variabilelor independente care nu conţine optimul.

Căutarea aleatoare (metoda Monte Carlo) constă în evaluarea funcţiei obiectiv într-un set de puncte generate pseudoaleator la fiecare iteraţie a algoritmului. Domeniul de explorare din jurul optimului aflat la fiecare iteraţie se restrânge, până când el devine mai mic decât cel impus; astfel, optimul de la ultima iteraţie se declară drept optim global.

Spre deosebire de metodele de gradient – care efectuează modificări *simultane* ce produc deplasări în spaţiul *n*-dimensional – metodele ***de căutare unidimensională (Gauss)*** se fondează pe modificarea *succesivă* a componentelor vectorului $\underline{x}$. Aceste metode se mai numesc şi metode ***de relaxare*** sau de *optimizare ciclică* de-a lungul axelor de coordonate (engl. *cyclic coordinate search*). Căutarea multidimensională este astfel transformată într-o

succesiune de căutări unidimensionale, fără a prospecta direcţia înaintării, ci doar prin relaxarea rând pe rând a tuturor direcţiilor axelor de coordonate. Există mai mulţi algoritmi bazaţi pe metoda relaxării, care diferă după modul în care se face varierea pasului de căutare.

Metodele de relaxare sunt în general mai lent convergente decât metodele obişnuite de gradient, fiindcă pot conţine mai multe iteraţii de căutare unidimensională, în funcţie de alegerea punctului de start şi a pasului. Ele pot deveni ineficiente sau cel puţin foarte lent convergente dacă funcţia obiectiv prezintă o vale (sau o creastă) care nu are direcţia paralelă cu axele de coordonate.

Se prezintă mai jos etapele unui algoritm de relaxare [CEAN 84], în care pasul este menţinut constant pe durata unui ciclu de explorare a tuturor direcţiilor, după care pasul este micşorat în progresie geometrică de raţie r, până când devine mai mic decât un prag dat, ε.

1. Se alege un punct iniţial caracterizat de vectorul $\underline{x}^{(0)} = \begin{bmatrix} x_1^{(0)} & x_2^{(0)} & \dots & x_n^{(0)} \end{bmatrix}$, valoarea iniţială a pasului de căutare, p, şi pragul ε.

2. Se iniţializează indexul coordonatei de relaxat: $k \leftarrow 1$.

3. Se iniţializează contorul de iteraţii din optimizarea în raport cu coordonata k: $i \leftarrow 0$.

4. La o iteraţie i se relaxează coordonata de index k, obţinându-se vectorul:

$$\underline{x}^{(i+1)} = \begin{bmatrix} x_1^{(i)} & \dots & \underbrace{x_k^{(i)} + p}_{x_k^{(i+1)}} & \dots & x_n^{(i)} \end{bmatrix}$$

Dacă $f\left(\underline{x}^{(i+1)}\right) \le f\left(\underline{x}^{(i)}\right)$, *atunci* $i \leftarrow i+1$ şi se reia etapa 4;

altfel *dacă* i=1 (prima iteraţie), *atunci* se face relaxarea în sens opus:

$$\underline{x}^{(i+1)} = \begin{bmatrix} x_1^{(i)} & \dots & x_k^{(i+1)} - 2p & \dots & x_n^{(i)} \end{bmatrix}$$

$p \leftarrow -p$

$i \leftarrow i+1$

şi se reia etapa 4;

altfel

dacă $f\left(\underline{x}^{(i)}\right) \le f\left(\underline{x}^{(i-1)}\right)$, *atunci* valoarea de optim a coordonatei k este:

$x_{k_{opt}} = x_k^{(i-1)}$

$k \leftarrow k+1$

şi se reia etapa 3 (se relaxează următoarea coordonată, k+1).

5. Se modifică pasul p, înmulţindu-l cu raţia r<1: $p \leftarrow p \cdot r$.

Dacă $p \le \varepsilon$, *atunci* s-a obţinut extremul funcţiei, STOP; *altfel* se reia de la etapa 2.

Funcţia Matlab de mai jos implementează algoritmul anterior pentru o funcţie scop de o variabilă vectorială de o dimensiune oarecare (numele fişierului ce conţine funcţia scop este transmis ca parametru de intrare, iar valorile funcţiei se evaluează cu `feval`).

```
function [x_opt,nr_it]=opt_rlx(fct_scop,x0,pas_init,r,tol)
%optimizare parametrică prin METODA RELAXĂRII

%fct_scop - şirul de caractere ce desemnează funcţia de optimizat
```

```
%x0        – punctul de start (vector de dimensiunea variabilei funcţiei de optimizat, n)
%pas_init  – pasul iniţial de căutare
%r         – raţia subunitară de modificare a pasului după fiecare ciclu de relaxare a tuturor
%            celor n coordonate
%tol       – valoarea de prag a pasului, când căutarea se opreşte

%iniţializări
   i=1;x(i,:)=x0;p=pas_init;
   n=length(x0);

   while (p>tol),
    p0=p*r;p=p0;
    k=1;d=zeros(1,n);
    stop_global=0;inapoi=0;
    while (~stop_global),
     d(k)=1;j=0;
     stop=0;
     while (~stop)
      if inapoi,
         p=p/2;
         inapoi=0;
      end;
      x(i+1,:)=x(i,:)+p*d;
      j=j+1;
      if feval(fct_scop,x(i+1,:))>feval(fct_scop,x(i,:)),
         if j==1,
            p=-2*p;
            i=i+1;
            x(i+1,:)=x(i,:)+p*d;
            inapoi=1;
         else stop=1;
              x(i+1,:)=[];
              i=i-1;
              d(k)=0;
              k=k+1;
              p=p0;
         end;
      end;
    i=i+1;
    end;
    if k<=n,
       stop_global=0;
    else stop_global=1;
    end;
    end;
```

```
    end;

    [nr_it n]=size(x);
    x_opt=x(nr_it,:);
```

Pentru exemplificare, s-a folosit o funcţie de două argumente reale:

```
function y=f(x)
    y=4+6*x(1)-4*x(2)+x(1)^2+2*x(2)^2-...
                    6*x(1)*x(2)+x(1)^4+2*x(1)^2*x(2);
```

care descrie o suprafaţă (figura 2.9), şi care are trei minime locale, calculabile analitic, situate aproximativ în punctele de coordonate (0;1), (0.3117;1.419) şi (–4.8117;–17.794).

Cele două apeluri de mai jos arată că, din acelaşi punct de start, (–1;3), dar cu alt pas iniţial (p=1, respectiv p=0.5), metoda relaxării poate furniza rezultate semnificativ diferite.

```
[x_opt,nr_it]=opt_rlx('f',[-1 3],1,0.5,1e-6)
x_opt =
    0.0000    1.0000
nr_it =
    84
[x_opt,nr_it]=opt_rlx('f',[-1 3],0.5,0.5,1e-6)
x_opt =
    0.3118    1.4191
nr_it =
   407
```

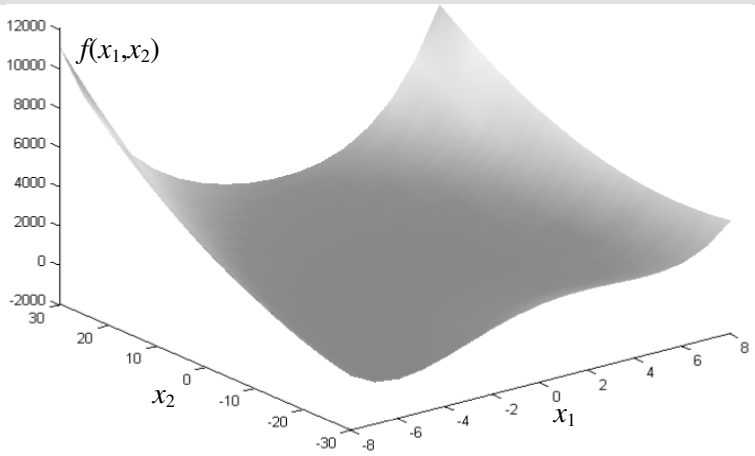

Fig. 1 O suprafaţă cu trei minime locale:

$$f(x_1, x_2) = 4 + 6x_1 - 4x_2 + x_1^2 + 2x_2^2 - 6x_1x_2 + x_1^4 + 2x_1^2x_2$$

Dintr-un alt punct de start se poate obţine un alt punct de minim local:

```
[x_opt,nr_it]=opt_rlx('f',[-3 -7],2,0.5,1e-5)
x_opt =
   -4.8158  -17.8200
nr_it =
        3577
```