

Gabriela Ciuprina, Daniel Ioan, Irina Munteanu, Mihai Rebican,
Radu Popa

Catedra de Electrotehnică, Universitatea “Politehnica” din București

Optimizarea numerică a dispozitivelor electromagnetice

Printech
București, 2002

Prefață

Această carte a fost concepută inițial ca fiind materialul didactic asociat cursului și laboratorului de *Metode numerice* dedicat studenților anului V al Facultății de Electrotehnică din Universitatea "Politehnica" București, de la specializarea *Inginerie electrică asistată de calculator*. Din motive care ne rămân neînțelese, acest curs nu mai face parte acum din programa școlară. Cartea este acum folosită ca suport didactic pentru disciplina *Metode numerice în ingineria electrică* dedicat studenților anului VI al Facultății de Electrotehnică din Universitatea "Politehnica" București, de la specializarea *Proiectarea și analiza asistată de calculator a dispozitivelor electromagnetice*. Lucrarea poate fi utilă tuturor studenților și inginerilor care doresc să se familiarizeze cu conceptele de bază și cu principalele metode numerice de optimizare utilizate în proiectarea asistată de calculator a dispozitivelor electromagnetice.

Cartea este structurată în 11 capitole și 6 anexe. Fiecare capitol are ca scop ilustrarea metodelor și algoritmilor de optimizare numerică dar și crearea și perfecționarea abilității de utilizare eficientă a calculatorului în inginerie. De aceea, materialul prezentat oferă comentarii parțiale legate de capitolele abordate și cuprinde în plus un număr de exerciții prin a căror rezolvare cititorul va putea înțelege în profunzime un anumit capitol.

Iată o scurtă descriere a cărții.

Primul capitol are ca scop familiarizarea cu concepte legate de optimizare cum ar fi: restricții, funcție obiectiv, extreme locale sau globale, puncte staționare, domeniu admisibil, optimizare vectorială, unimodalitate și convexitate.

Minimizarea unidimensională (a funcțiilor reale care depind de o singură variabilă reală) este o cărămidă de bază a minimizării multidimensionale și de aceea capitolul al doilea propune testarea și analiza câtorva algoritmi de căutare de ordinul zero pentru minimizare unidimensională.

Capitolul al treilea tratează cazul unidimensional al funcțiilor care prezintă un anumit grad de netezime. Pentru acestea se pot aplica tehnici mai eficiente – care găsesc minimul într-un număr mai mic de pași – dar care, pe de altă parte, necesită și evaluări ale derivatei funcției. Tehnicile din această categorie se bazează în general pe aproximarea locală a

profilului funcției cu o funcție polinomială de grad mic, căreia i se poate determina ușor minimumul. Algoritmii prezentați în această lucrare sunt de ordinul unu (necesită evaluarea primei derivate) și aparțin clasei metodelor de aproximare polinomială.

Al patrulea capitol urmărește aplicarea metodelor de optimizare descrise până acum pentru dimensionarea unui magnet permanent, astfel încât indicele său de calitate să fie maxim.

În capitolele cinci și șase sunt prezentați doi dintre cei mai celebri algoritmi determiniști de ordin zero (care nu au nevoie de calculul derivatelor funcției obiectiv) folosiți pentru minimizarea funcțiilor de mai multe variabile. Prima metodă prezentată este metoda simplexului descendent, cunoscută și sub numele de metoda lui Nelder și Mead. A doua metodă deterministă de ordin zero folosită pentru minimizarea multidimensională este metoda Powell. Deosebirea principală dintre cele două metode prezentate este aceea că metoda simplexului descendent nu are nevoie explicită de un algoritm de minimizare unidimensională ca parte a strategiei de calcul, așa cum are nevoie metoda Powell.

Capitolul șapte se referă la metode deterministe de ordinul unu. Metodele deterministe de ordinul unu determină minimumul funcțiilor multidimensionale unimodale fără restricții folosind vectorul gradient. Aceste metode sunt cunoscute sub numele de metode de tip gradient. Dacă derivatele sunt continue și pot fi evaluate analitic, metodele de tip gradient sunt mai eficiente decât metodele de căutare de ordinul zero, cum ar fi metoda simplex, care folosesc numai evaluări ale funcției obiectiv. Metodele de tip gradient sunt recomandate pentru funcții cu derivate ușor de calculat analitic. În acest capitol sunt prezentate: metoda celei mai rapide coborri (“steepest descendent” cunoscută și sub numele de metoda gradientului) și metoda gradientilor conjugați.

Din clasa metodelor deterministe de ordinul unu fac parte și metodele numite quasi-Newton. Ele sunt numite astfel deoarece încearcă să simuleze iterații de tip Newton-Raphson, plasându-se cumva între metoda gradientului și metoda Newton. Metoda Newton necesită evaluarea inversei matricei Hessian, lucru care este foarte costisitor din punct de vedere numeric. Ca urmare, a apărut ideea de a lucra cu o aproximare a inversei matricei Hessian calculată cu ajutorul vectorului gradient evaluat în iterațiile precedente, idee care stă la baza metodelor quasi-Newton. Variantele metodelor de tip quasi-Newton diferă prin felul în care se face această aproximare. Aproximările pot fi din cele mai simple, în care matricea aproximativă rămâne constantă pe parcursul iterațiilor, până la cele mai avansate, în care se construiesc aproximații din ce în ce mai bune ale inversei matricei Hessian, pe baza informațiilor adunate în timpul procesului de coborâre. Această din urmă abordare corespunde metodelor din clasa algoritmilor de metrică variabilă. Prima și una din cele mai importante scheme de construcție a inversei matricei Hessian a fost propusă de Davidon (1959). Metoda a fost mai târziu modificată și îmbunătățită de

Fletcher și Powell (1964), algoritmul propus de ei fiind cunoscut sub numele de algoritmul Davidon-Fletcher-Powell. O altă variantă este cunoscută sub numele Broyden-Fletcher-Goldfarb-Shanno (BFGS). Algoritmii DFP și BFGS diferă numai în detalii legate de eroarea de rotunjire, toleranțele de convergență și alte aspecte de acest tip. Totuși, a devenit în general recunoscut că, empiric, schema BFGS este superioară din punct de vedere al acestor detalii. Capitolul opt prezintă acești doi algoritmi.

De cele mai multe ori se dorește însă găsirea unui extrem global și nu numai a unui local, lucru ce pretinde explorarea întregului domeniu de căutare și nu numai o vecinătate a inițializării. Pentru a determina un extrem global se poate proceda astfel: se execută algoritmul determinist pentru mai multe puncte inițiale de căutare împrăștiate uniform în domeniul de căutare și apoi se alege dintre soluțiile găsite valoarea cea mai bună sau se perturbă un extrem local găsit pentru a vedea dacă algoritmul determinist cu această inițializare regăsește același extrem. Ca o alternativă la aceste două abordări, au început să fie folosiți tot mai des algoritmi stocastici. Aceștia nu garantează găsirea unui extrem global, dar ei au o probabilitate mult mai mare de a găsi un astfel de extrem. De asemenea ei mai au avantajul că nu necesită evaluarea derivatelor funcției de optimizat, fiind în consecință algoritmi de ordinul zero. Dezavantajul lor este acela că numărul de evaluări de funcții necesar pentru găsirea optimului este relativ mare față de cazul metodelor deterministe, dar în multe situații acest sacrificiu trebuie făcut, pentru că acești algoritmi sunt singurii care dau rezultate numerice acceptabile. În acest scop, capitolul nouă prezintă câteva dintre metodele stocastice de căutare a unui optim și anume metoda căutării aleatoare (sau a drumului aleator) în două variante precum și algoritmi genetici.

Capitolul zece urmărește aplicarea metodelor de optimizare prezentate în capitolele anterioare pentru optimizarea dispozitivului de producere a câmpului magnetic uniform, cunoscut sub numele de *bobinele lui Helmholtz*.

În încercarea de a rezolva o problemă de optimizare cu calculatorul trebuie plecat de la observația, unanim acceptată, că nu există un program pentru calculator (cod) general, capabil să rezolve eficient orice problemă de optimizare neliniară. Datorită diversității atât a problemelor de optimizare, a algoritmilor cți și a programelor de calculator disponibile, alegerea codului potrivit pentru o problemă concretă este dificilă și cere experiență în optimizări dar și înțelegerea profundă a problemei de rezolvat.

În viața reală a cercetării științifice și ingineriei, rareori se inventează un algoritm original de optimizare și un program complet nou pentru rezolvarea unei probleme concrete, ci cel mai adesea se refolosesc coduri existente, asigurându-se în acest fel eficiența profesională în rezolvarea problemelor. În această activitate, experiența căpătată în parcurgerea capitolelor prezentate anterior este de un real folos.

Indiferent dacă se folosesc programe de optimizare existente sau se dezvoltă coduri noi, se recomandă cu tărie ca acestea să fie verificate folosind probleme și modele de test,

de preferință cât mai apropiate de problema concretă de rezolvat.

De aceea, scopul ultimului capitol este de a familiariza cititorul cu principalele abordări folosite în rezolvarea cu tehnici profesionale, bazate pe reutilizarea software, a problemelor de optimizare. Spre deosebire de capitolele anterioare în care accentul era pus pe anatomia algoritmilor de optimizare, în această capitol atenția este focalizată asupra modului în care pot fi folosite programe existente în rezolvarea unor probleme noi.

Anexele oferă informații suplimentare legate de tipurile de probleme de optimizare (anexa A), funcțiile de test pentru algoritmi de optimizare (anexa B), metodele deterministe (anexa D) sau stocastice (anexa E) de optimizare. Sunt prezentate și două definiții de probleme de test, propuse de COMPUMAG Society, ce vizează optimizarea dispozitivelor electromagnetice (anexa C). Pentru cei care nu sunt familiarizați cu programul Scilab, anexa F oferă o scurtă introducere în utilizarea acestui pachet.

Contribuția autorilor la realizarea lucrării este următoarea:

- S.l.dr.ing. Gabriela Ciuprina a conceput, tehnoredactat capitolele 1, 2, 4, 5, 8, 9, 10, anexele A, B, C, F și subparagrafele 1 și 2 ale anexei D, și a realizat integrarea finală a lucrării.
- Prof.dr.ing. Daniel Ioan a conceput capitolul 11 și anexele D și E.
- Conf.dr.ing. Irina Munteanu a conceput și tehnoredactat capitolul 6;
- As.ing. Mihai Rebican a conceput și tehnoredactat capitolul 7;
- As.dr.ing. Radu Popa a conceput și tehnoredactat capitolul 3.

Autorii mulțumesc doamnei ing. Suzana Jerpelea pentru ajutorul acordat în realizarea programelor aferente capitolului 5.

În cele din urmă, dar nu în ultimul rând autorii sunt recunoscători referenților științifici: prof.dr.ing.F.M.G. Tomescu și prof.dr.ing. Mihai Iordache pentru amabilitatea de a citi cu foarte multă răbdare și atenție acest material. Observațiile lor sunt incluse în această variantă a lucrării. Deosebit de calde mulțumiri sunt aduse și domnului prof.dr.ing. Corneliu Popeea pentru sugestiile și observațiile extrem de valoroase pe care ni le-a oferit pe parcursul conceperii acestui material.

Alte comentarii și sugestii în vederea îmbunătățirii acestei cărți sunt binevenite la oricare din adresele gabriela@lmn.pub.ro, daniel@lmn.pub.ro, irina@lmn.pub.ro, mihai_r@lmn.pub.ro.

Cuprins

1	Introducere în problema optimizării	1
1.1	Formularea problemei	1
1.1.1	Minimizare	1
1.1.2	Minime locale și globale	2
1.1.3	Gradienți	2
1.1.4	Puncte staționare	3
1.1.5	Restricții	3
1.1.6	Convergență	4
1.1.7	Probleme de optimizări vectoriale	4
1.2	Optimizarea unidimensională	5
1.2.1	Funcții unimodale	5
1.2.2	Funcții convexe	6
1.3	Optimizare în n dimensiuni	7
1.3.1	Curbe de nivel	7
1.3.2	Unimodalitate și convexitate	9
2	Minimizări unidimensionale - metode de căutare	11
2.1	Formularea problemei	11
2.2	Metoda rețelei	12
2.2.1	Principiul metodei	12
2.2.2	Algoritmul metodei	12

2.2.3	Acuratețe și efort de calcul	13
2.3	Metoda Fibonacci	13
2.3.1	Principiul metodei	14
2.3.2	Algoritmul metodei	16
2.3.3	Acuratețe și efort de calcul	17
2.4	Metoda secțiunii de aur	18
2.4.1	Principiul metodei	18
2.4.2	Algoritmul metodei	19
2.4.3	Acuratețe și efort de calcul	20
2.4.4	Comparație cu metoda Fibonacci	20
3	Minimizări 1D - metode de aproximare	23
3.1	Formularea problemei	24
3.2	Metoda falsei poziții (aproximării parabolice)	24
3.2.1	Principiul metodei	24
3.2.2	Algoritmul metodei	26
3.2.3	Efort de calcul	27
3.3	Metoda aproximării cubice	27
3.3.1	Principiul metodei	27
3.3.2	Algoritmul metodei	28
3.3.3	Acuratețe și efort de calcul	29
4	Minimizări 1D - aplicație	31
4.1	Descrierea problemei de optimizare	31
4.2	Formularea problemei de optimizare	33
4.2.1	Aproximarea curbei de material cu ajutorul unei expresii analitice	34
4.2.2	Aproximarea liniară pe porțiuni a curbei de material	35
4.3	Proiectarea circuitelor magnetice cu magneti permanenți	35

5	Metoda simplexului descendent	37
5.1	Formularea problemei	38
5.2	Probleme de test	38
5.2.1	Cămila cu șase cocoase	39
5.2.2	Funcția lui Rosenbrock (“banana”)	40
5.3	Metoda simplexului descendent	41
5.3.1	Principiul metodei	41
5.3.2	Algoritmul metodei	42
5.3.3	Exerciții	44
6	Metoda Powell	47
6.1	Formularea problemei	47
6.2	Minimizarea după o direcție a funcțiilor de mai multe variabile	47
6.3	Principiul metodei Powell	51
6.4	Îmbunătățiri ale metodei Powell	55
6.5	Viteza de convergență	58
7	Metoda gradientilor conjugați	61
7.1	Formularea problemei	61
7.2	Metoda celei mai rapide coborri (metoda gradientului)	62
7.2.1	Principiul metodei	62
7.2.2	Algoritmul metodei	66
7.2.3	Efort de calcul	67
7.3	Metoda gradientilor conjugați	69
7.3.1	Principiul metodei	69
7.3.2	Algoritmul metodei	71
7.3.3	Efort de calcul	71
7.4	Aspecte legate de convergență	72

8 Metode quasi-Newton	75
8.1 Metoda Newton modificată	76
8.2 Construcția inversei matricei Hessian. Corecția de rangul unu.	77
8.3 Metoda Davidon-Fletcher-Powell	79
8.4 Clasa de metode Broyden	81
8.5 Metode de metrică variabilă sau gradienti conjugați?	84
9 Metode stocastice de optimizare	87
9.1 Metoda căutării aleatoare (drumului aleator)	88
9.1.1 Varianta Matyas	88
9.1.2 Varianta îmbunătățită	89
9.2 Programe evoluționiste. Algoritmi genetici.	90
9.2.1 Structura unui program evoluționist	91
9.2.2 Algoritmi genetici	93
10 Aplicație - Bobinele Helmholtz	101
10.1 Bobinele Helmholtz	101
10.1.1 Descrierea dispozitivului	101
10.1.2 Considerații teoretice	101
10.2 Formularea problemei de optimizare	104
10.2.1 Funcție obiectiv de tip minimax	105
10.2.2 Funcție obiectiv de tip norma Euclidiană	106
11 Software profesional pentru optimizare	107
11.1 Abordări profesionale ale problemelor de optimizare	108
11.2 Surse pentru software destinat optimizării	109
11.3 Rezolvarea interactivă a problemelor simple	112
11.4 Utilizarea rutinelor din biblioteci matematice	114
11.5 Utilizarea serviciului de optimizare prin Internet	114

A	Tipuri de probleme de optimizare	117
A.1	Optimizări scalare	117
A.2	Optimizări vectoriale	120
A.2.1	Optimalitate Pareto	120
A.2.2	Stabilirea funcției obiectiv	121
B	Funcții de test pentru algoritmi de optimizare	123
B.1	Probleme care au doar restricții de domeniu	123
B.2	Probleme de optimizare cu restricții	130
C	Exemple de probleme de optimizare a dispozitivelor electromagnetice	135
C.1	Dispozitiv de stocare a energiei magnetice	135
C.2	Matrită cu electromagnet	137
D	Metode deterministe pentru optimizarea locală	143
D.1	Metode de optimizare deterministe pentru probleme fără restricții	143
D.2	Tratarea restricțiilor	145
D.3	Optimizare neliniară fără restricții	149
D.4	Optimizare neliniară cu restricții	152
D.5	Problema celor mai mici pătrate neliniare	156
D.6	Rezolvarea sistemelor de ecuații neliniare	158
D.7	Programare liniară	160
D.8	Programarea pătratică	163
D.9	Arborele de decizie pentru metoda deterministă	164
E	Metode stocastice pentru optimizarea globală	167
E.1	Algoritmul genetic canonic	167
E.2	Strategii evoluționiste	171
E.3	Prototipul unui algoritm evoluționist	173
E.4	Algoritmi cu nișe pentru optimizarea funcțiilor multimodale	174

E.5	Algoritmi evoluționiști paraleli	178
E.6	Analiza parametrilor algoritmilor evoluționiști	179
F	Inițiere în Ψlab (<i>Scilab</i>)	181
F.1	Înainte de toate....	181
F.2	Variabile și constante.	182
F.3	Atribuirii și expresii	183
F.4	Generarea vectorilor și matricelor	186
F.5	Instrucțiuni grafice	189
F.6	Programare în <i>Scilab</i>	190
F.6.1	Editarea programelor	190
F.6.2	Operații de intrare/ieșire	191
F.6.3	Structuri de control	192
F.6.4	Funcții	194
F.7	Alte facilități de postprocesare	195
	Bibliografie și webografie	199

Capitolul 1

Introducere în problema optimizării

Scopul acestui capitol este de a vă familiariza cu câteva din principalele concepte legate de optimizare cum ar fi: restricții, funcție obiectiv, extreme locale sau globale, puncte staționare, domeniu admisibil, optimizare vectorială, unimodalitate și convexitate.

1.1 Formularea problemei

1.1.1 Minimizare

Problema de bază a optimizării constă în găsirea minimului unei mărimi scalare E care depinde de n parametri notați x_1, x_2, \dots, x_n . Dacă E depinde de x_1, x_2, \dots, x_n prin intermediul unei funcții $f : \Omega \rightarrow \mathbb{R}$, atunci se scrie pe scurt:

$$\text{minimizează } E = f(x_1, x_2, \dots, x_n), \quad (1.1)$$

înțelegându-se că se cer valorile variabilelor independente $(x_1, x_2, \dots, x_n) = \operatorname{argmin} f$ din domeniul de definiție pentru care f are cea mai mică valoare, care va fi și ea determinată. Dacă domeniul de definiție se extinde în întreg spațiul $\Omega = \mathbb{R}^n$, atunci problema de optimizare este **fără restricții**, în caz contrar ea este una **cu restricții**.

Exercițiul 1.1: Problema optimizării a fost formulată ca o problemă de minimizare. Este aceasta o limitare a definiției? Cum se formulează problema maximizării?

De obicei E înglobează criteriile de proiectare și/sau economie într-un singur număr care adesea măsoară prețul de producție sau diferența între performanța cerută și performanța obținută. Din acest motiv, funcției f i se mai spune **funcție de cost** sau **funcție obiectiv**.

Mulțimea celor n parametri va fi reprezentată ca un vector coloană cu componente reale:

$$\mathbf{x} = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^n. \quad (1.2)$$

Vom nota cu $\mathbf{x}_{\min} = \operatorname{argmin} f(x)$, punctul $\mathbf{x} \in \Omega$ pentru care f este minimă, iar valoarea minimă a funcției o vom nota cu: $E_{\min} = \min\{f(x) : x \in \Omega\}$.

1.1.2 Minime locale și globale

Dacă E_{\min} este cea mai mică valoare posibilă a lui E pentru orice \mathbf{x} din domeniul de definiție Ω , atunci se spune că punctul \mathbf{x}_{\min} este un punct de **minim global**. Acesta în general s-ar putea să nu fie unic. Dacă punctul găsit reprezintă un minim doar într-o vecinătate a sa, se spune în acest caz că este un **minim local**. În practică este dificil de stabilit dacă un minim găsit este global sau numai local.

Exercițiul 1.2: *Dați exemplu de o funcție f care să aibă și minime locale și minime globale.*

1.1.3 Gradienti

Unele metode de optimizare necesită informații legate de derivatele parțiale de ordinul unu sau doi ale funcției obiectiv în raport cu parametrii de optimizare.

Vectorul gradient Jacobian, notat \mathbf{g} , este definit (atunci când f este diferențiabilă) ca fiind transpusa vectorului gradient ∇f , care este un vector linie în care apar derivatele parțiale de ordinul 1:

$$\mathbf{g} = (\nabla f)^T = \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \dots \frac{\partial f}{\partial x_n} \right]^T. \quad (1.3)$$

Gradientul este un câmp vectorial normal la oricare dintre hipersuprafețele de nivel $f(x_1, x_2, \dots, x_n) = \text{constant}$.

Matricea pătrată, simetrică, de dimensiune n , care conține derivatele parțiale de ordinul doi ale funcției f este cunoscută sub numele de **matricea Hessian**, și este notată cu \mathbf{H} :

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \dots & & \\ \dots & & & \\ \dots & & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \dots & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \quad (1.4)$$

Exercițiul 1.3: *Calculați expresiile Jacobianului și Hessianului funcției $f(x, y, z) = x^2 + 2y^3 + 7z^4$. Evaluați aceste matrice în punctul $x = y = z = 0$.*

1.1.4 Puncte staționare

O condiție necesară pentru punctele de extrem local se poate formula cu ajutorul noțiunii de punct staționar. Prin definiție, punctele în care gradientul ∇f se anulează, adică zerourile sistemului:

$$\begin{aligned} \frac{\partial f}{\partial x_1}(x_1, x_2, \dots, x_n) &= 0 \\ &\dots\dots\dots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \dots, x_n) &= 0 \end{aligned} \tag{1.5}$$

se numesc **puncte staționare** (sau critice) ale funcției f (presupusă diferentiabilă).

Exercițiul 1.4: Ce semnificație geometrică au punctele staționare în cazurile $n = 1$ și, respectiv $n = 2$?

Un rezultat important asociat problemei găsirii punctelor de extrem este **teorema lui Fermat**: Dacă f este o funcție diferentiabilă și \mathbf{x}_0 este un punct de extrem local, atunci \mathbf{x}_0 este un punct staționar al lui f .

Exercițiul 1.5: Ce semnificație geometrică are teorema lui Fermat în cazul $n = 1$?

1.1.5 Restricții

În problemele practice de optimizare, există anumite restricții impuse parametrilor, acestea restrângând domeniul admisibil de căutare a minimumului. O restricție des întâlnită este **restricția de domeniu** impusă variabilelor x_i , care este de forma:

$$x_{L,i} \leq x_i \leq x_{U,i}, \tag{1.6}$$

unde $x_{L,i}$ și $x_{U,i}$ reprezintă limite fixate, inferioară, respectiv, superioară. Aceste limite definesc o formă paralelipipedică a domeniului de definiție Ω . În general, există **restricții de tip inegalitate** exprimate cu ajutorul funcțiilor $g_i : \Omega \rightarrow \mathbb{R}$:

$$g_i(x_1, x_2, \dots, x_n) \leq 0. \tag{1.7}$$

Este de asemenea posibil ca între parametrii de optimizat să existe și **restricții de tip egalitate**:

$$h_j(x_1, x_2, \dots, x_n) = 0. \tag{1.8}$$

Problema minimizării cu restricții are forma generală:

$$\min\{f(x) : g_i(x) \leq 0, i \in I; h_j(x) = 0, j \in J\}, \tag{1.9}$$

în care I și J sunt mulțimi de indici.

Exercițiul 1.6: *Cum pot fi exprimate restricțiile de domeniu sub forma restricțiilor de tip inegalitate?*

Domeniul de căutare în care restricțiile sunt satisfăcute se numește **regiune admisibilă**. Aceasta este partea din Ω în care sunt satisfăcute relațiile (1.7) și (1.8).

Optimizarea cu restricții este mult mai dificilă decât cea fără restricții. De multe ori, problemele de optimizare cu restricții sunt reformulate astfel încât ele să se reducă la rezolvarea unor probleme de optimizare fără restricții.

1.1.6 Convergență

Pentru a compara diferite tehnici iterative de optimizare la care soluția numerică de la iterația k se calculează în funcție de cea de la iterația $k + 1$, trebuie găsite răspunsuri la următoarele două întrebări:

1. Este convergentă procedura către un minim și este acest minim unul global?
2. Care a fost rata (viteza) de convergență?

Valoarea E_k a funcției obiectiv în timpul iterațiilor este în majoritatea cazurilor singura mărime care reflectă progresul algoritmului de minimizare. Dacă E_k nu mai scade un număr de iterații, este posibil să se fi atins un minim. Totuși, este aproape imposibil să se prezică dacă acest minim este unul global. Nici una din tehnicile iterative de optimizare nu garantează convergența către un minim global.

Viteza relativă de convergență este de obicei estimată prin numărul de evaluări ale funcției f necesare pentru a reduce valoarea E_k de un anumit număr de ori.

1.1.7 Probleme de optimizări vectoriale

În majoritatea problemelor, optimizarea înseamnă satisfacerea mai multor cerințe (de exemplu: cost minim, randament maxim, solicitări minime), aceasta însemnând că multe funcții obiectiv $f_i(\mathbf{x})$, $i = 1, \dots, m$ trebuie minimizate simultan. Problema optimizării multiobiectiv poate fi redusă la una de tipul (1.1) folosind de exemplu minimizarea în sensul celor mai mici pătrate, sau un criteriu de tip “minimax”.

Minimizarea în sensul **celor mai mici pătrate** constă în minimizarea expresiei:

$$\text{minimizează } E = \sum_{i=1}^m (w_i f_i(\mathbf{x}))^2, \quad (1.10)$$

în care w_i se numesc **ponderi** sau **factori de penalizare** al căror scop este de a pondera importanța diferitor obiective de minimizat.

Criteriul de tip **minimax** constă în minimizarea normei Cebîșev, respectiv a celei mai mari valori pentru modul:

$$\text{minimizează } E = \max |w_i f_i(\mathbf{x})|. \quad (1.11)$$

Dezavantajul acestei din urmă abordări îl constituie faptul că E este de obicei o funcție care nu este netedă, neputându-se deci defini derivatele ei în raport cu parametrii de optimizare.

1.2 Optimizarea unidimensională

Optimizarea unidimensională corespunde cazului $n = 1$, problema (1.1) simplificându-se la:

$$\text{minimizează } E = f(x), \quad (1.12)$$

cu $x \in [x_L, x_U] \subset \mathbb{R}$. Optimizarea reprezintă deci o căutare a minimului într-un interval $\Omega = [x_L, x_U]$. Foarte multe proceduri de minimizare multidimensională se reduc la o secvență de optimizări unidimensionale, și de aceea această problemă trebuie rezolvată eficient și cu acuratețe.

1.2.1 Funcții unimodale

O funcție f de un singur argument se numește **unimodală** atunci când are un singur minim în domeniul ei de definiție, deci x_{\min} este singura valoare a lui x pentru care $f(x) \leq f(y)$ pentru orice y în domeniul de definiție $[x_L, x_U]$. Această definiție nu face referire la derivatele funcției și de aceea ea poate fi aplicată atât funcțiilor continue cât și celor discontinue.

Exercițiul 1.7: Pentru funcțiile reprezentate grafic în figurile 1.1 ÷ 1.4 stabiliți dacă sunt continue sau nu, respectiv dacă sunt unimodale sau nu.

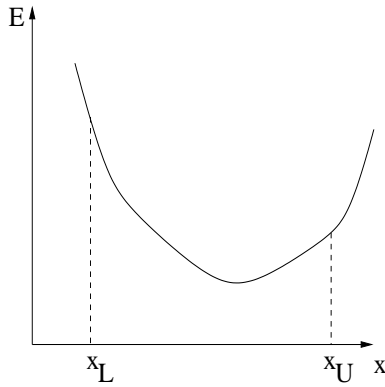


Figura 1.1: Vezi exercițiul 1.7 .

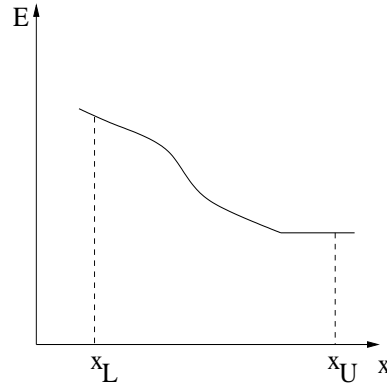


Figura 1.2: Vezi exercițiul 1.7 .

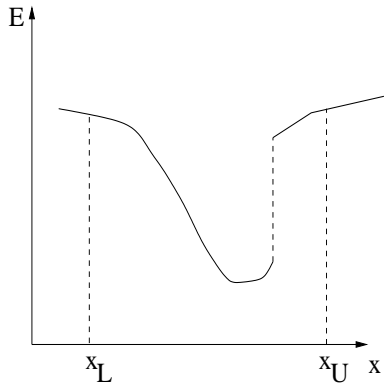


Figura 1.3: Vezi exercițiul 1.7 .

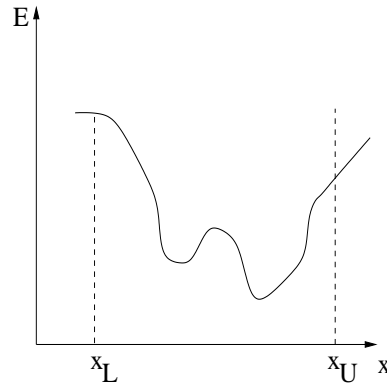


Figura 1.4: Vezi exercițiul 1.7 .

1.2.2 Funcții convexe

O funcție continuă este **convexă** dacă, pentru orice x și y din domeniul de definiție și orice λ cuprins între 0 și 1, are loc inegalitatea:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y). \quad (1.13)$$

Dacă inegalitatea este strictă, funcția se numește **strict convexă**. Funcția f se numește **concavă** dacă $-f$ este convexă.

Dacă o funcție este convexă, atunci minimul local este de asemenea global.

Geometric, o funcție este strict convexă dacă un segment de dreaptă care unește orice două puncte de pe grafic este situat deasupra graficului.

Exercițiul 1.8: Comentăți convexitatea funcțiilor reprezentate grafic în figurile 1.1 ÷ 1.4.

Exercițiul 1.9: Scrieți¹ un program Scilab care să deseneze graficul funcției:

$$E = -|\cos \pi x| + 10 \left(x - \frac{1}{4} \right)^2.$$

Stabiliți dacă această funcție este unimodală sau convexă în intervalul $[0,1]$.

1.3 Optimizare în n dimensiuni

Dacă o funcție reală de mai multe variabile reale este derivabilă de suficient de multe ori, ea poate fi dezvoltată în serie Taylor:

$$f(\mathbf{x} + \Delta \mathbf{x}) = f(\mathbf{x}) + \mathbf{g}^T \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^T \mathbf{H} \Delta \mathbf{x} + \dots, \quad (1.14)$$

unde \mathbf{g} reprezintă vectorul Jacobian iar \mathbf{H} reprezintă matricea Hessian.

Dacă derivatele de ordinul unu există, atunci într-un minim local vectorul gradient \mathbf{g} are toate componentele nule (conform teoremei lui Fermat). Dacă există și derivate de ordinul doi, matricea Hessian \mathbf{H} este pozitiv definită în punctul de minim.

1.3.1 Curbe de nivel

În cazul bidimensional ($n = 2$), reprezentarea grafică a funcției obiectiv prin curbe de nivel (izovalori) este foarte sugestivă. Fiecare curbă de nivel reprezintă mulțimea punctelor pentru care $f(\mathbf{x})$ este constantă.

Exercițiul 1.10: Figura 1.5 reprezintă curbe de nivel ale funcției de optimizat într-o problemă fără restricții, iar figura 1.6 pentru o problemă cu restricții.

- Comentați continuitatea funcțiilor reprezentate în aceste figuri.
- Caracterizați punctele A , B și C .

Exercițiul 1.11: Punctele staționare nu reprezintă neaparat puncte de extrem. Verificați că, pentru următoarele patru funcții, originea $(0,0)$ este un punct staționar:

$$\begin{aligned} f_1(x, y) &= xy & f_2(x, y) &= x^3 - 3xy^2 \\ f_3(x, y) &= x^2 & f_4(x, y) &= x^2 y^2 \end{aligned}$$

¹Câteva sfaturi pentru lucrul în LMN: În directorul \$HOME creați un director numit (de exemplu) MNIE, iar în acesta din urmă directoare numite tema1, tema2, etc. În directorul ~/MNIE/tema1 creați fișierele necesare rezolvării exercițiilor din această capitole.

Pentru rezolvarea acestui exercițiu, creați un fișier numit “functii.sci” în care scrieți definiția funcției și un fișier numit “main_1.9.sci” în care scrieți programul Scilab pentru exercițiul de mai sus. Pentru trasarea propriu-zisă a graficului folosiți funcția Scilab `fplot2d`.

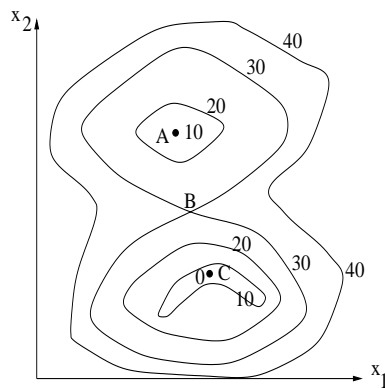


Figura 1.5: Vezi exercițiul 1.10

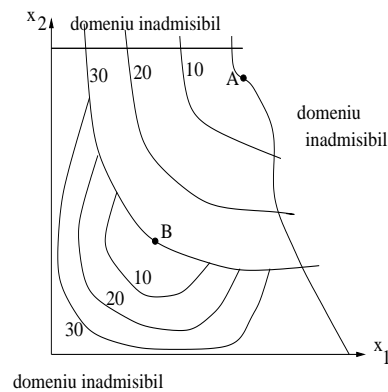


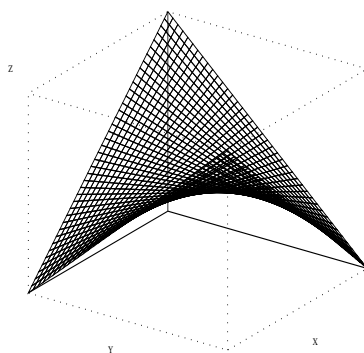
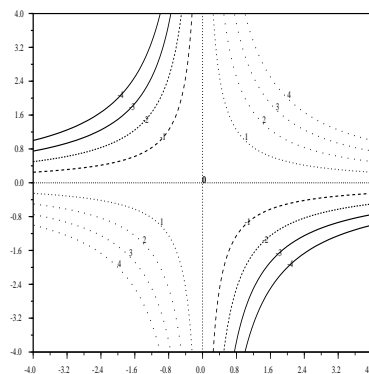
Figura 1.6: Vezi exercițiul 1.10

Pentru funcția f_1 se spune că originea este un **punct șa** deoarece există două direcții care trec prin origine, pentru una din direcții originea fiind un punct de maxim, iar pentru cealaltă, originea este un punct de minim. Figura 1.7 reprezintă graficul funcției f_1 iar figura 1.8 reprezintă curbele de nivel ale aceleiași funcții.

Pentru funcția f_2 , reprezentată în figurile 1.9 și 1.10, originea este un **punct șa maimuță**. Pentru funcția f_3 , originea este un **punct albie** iar pentru funcția f_4 originea este un **punct încrucișare de albie**.

a) Justificați aceste denumiri și caracterizați comportarea funcțiilor respective după mai multe direcții care trec prin origine.

b) Pentru cele patru funcții, stabiliți în care cazuri originea este punct de extrem.

Figura 1.7: Funcția f_1 de la exercițiul 1.11 .Figura 1.8: Curbe de nivel ale funcției f_1 .

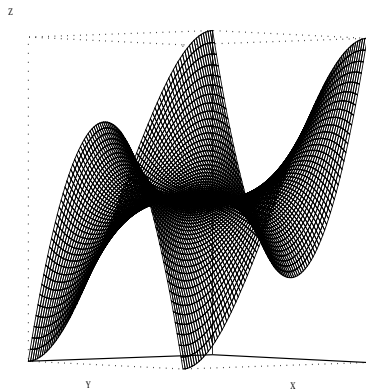


Figura 1.9: Funcția f_2 de la exercițiul 1.11 .

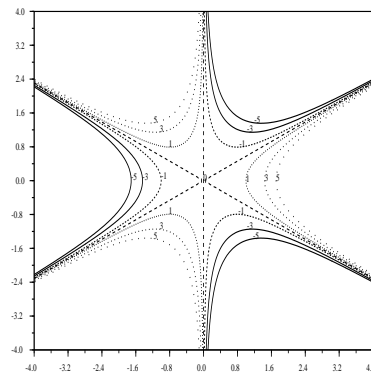


Figura 1.10: Curbe de nivel ale funcției f_2 .

Exercițiul 1.12: Scrieți un program Scilab care să deseneze curbele de nivel ale funcțiilor f_3 și f_4 definite la exercițiul anterior, precum și graficele lor. ²

1.3.2 Unimodalitate și convexitate

Exercițiul 1.13: Extindeți definițiile de unimodalitate și convexitate la cazul multi-dimensional.

Exercițiul 1.14: Scrieți un program Scilab care să deseneze curbele de nivel ale funcției obiectiv:

$$E = [10 - (x_1 - 2)^2 - (x_2 - 1)^2]^2 + (x_1 - 2)^2 + (x_2 - 4)^2$$

definită în regiunea $x_1 \in [-4, 6]$, $x_2 \in [-4, 6]$. Identificați punctele de minim și de maxim.

Exercițiul 1.15: a) Reluați exercițiul 1.14 presupunând că parametrii funcției sunt supuși restricției: $x_1 + x_2 - 3 \leq 0$.

b) Reluați punctul a în cazul în care există și restricția suplimentară: $x_2 = -1$.

² Indicație: Folosiți funcțiile Scilab `fcontour` și `fplot3d`.

Capitolul 2

Minimizări unidimensionale - algoritmi de ordinul zero bazați pe metode de căutare

Minimizarea unidimensională este o cărămidă de bază a minimizării multidimensionale. În cadrul acestui capitol se propune testarea și analizarea câtorva algoritmi de căutare de ordinul zero pentru minimizare unidimensională. Un **algoritm de optimizare** se spune că este **de ordinul zero** dacă el nu folosește derivate ale funcției de optimizat. Dacă un astfel de algoritm are nevoie de derivate, se spune că el este de ordinul unu, doi, etc. în funcție de ordinul derivatelor necesare.

Există două clase de metode disponibile pentru optimizarea unidimensională:

- Funcția de minimizat $f(x)$ este aproximată printr-o funcție cunoscută care poate fi analizată pentru a se determina minimumul. De obicei, funcția de aproximare este un polinom de grad mic. Aceste metode se numesc **metode de aproximare**.
- Intervalul care conține minimumul este micșorat, prin evaluarea funcției f în anumite puncte. Există câteva tehnici pentru plasarea acestor puncte. Astfel de metode se numesc **metode de căutare**.

2.1 Formularea problemei

Fie $f : [a, b] \rightarrow \mathbb{R}$ o funcție reală, continuă pe porțiuni. Se cere să se calculeze minimumul funcției f .

2.2 Metoda rețelei

2.2.1 Principiul metodei

Cea mai simplă metodă pentru determinarea aproximativă a minimumului global al funcției f (numită uneori “metoda rețelei”) constă în împărțirea intervalului $[a, b]$ în n subintervale egale prin punctele $x_i = a + i * h$, $i = 0, \dots, n$, $h = (b - a)/n$. Se calculează valorile $f(x_i)$ și se selectează valoarea minimă.

Evident, pentru a putea “prinde” toate variațiile semnificative ale funcției f se impune ca pasul h să fie suficient de mic.

2.2.2 Algoritmul metodei

Algoritmul acestei metode este următorul:

funcție met_retelei(**real** a, b , **funcție** f , **întreg** n)

$h = (b - a)/n$

$minim = f(b)$

pentru $i = 0 : n - 1$

$x = a + ih$

$val = f(x)$

dacă ($val < minim$)

$minim = val;$

întoarce $minim$

Exercițiul 2.1: a) Scrieți funcția Scilab de tipul:

`function rez = met_retelei(a,b,f,n)`

care implementează algoritmul de mai sus.

b) Scrieți într-un fișier definițiile următoarelor funcții:

$$f_1(x) = \frac{x^2}{2} \ln x - \frac{x^2}{4} - x$$

$$f_2(x) = \frac{\sqrt{\cos(x^2 - 1) + x^2}}{(3x^2 + 17)^{1/3}}$$

$$f_3(x) = \sqrt{0.005x + 100 \frac{20.1 + 2x}{20.1 + 400x}}$$

c) Scrieți un program Scilab care să determine minimele funcțiilor $f_1 : [1, 3] \rightarrow \mathbb{R}$, $f_2 : [0, 5] \rightarrow \mathbb{R}$, $f_3 : [20, 35] \rightarrow \mathbb{R}$. Pentru verificare, reprezentați graficele acestor funcții.

Exercițiul 2.2: Imaginați un exemplu de funcție (grafic), pentru care metoda rețelei (pentru un anumit număr de puncte), ar eșua.

2.2.3 Acuratețe și efort de calcul

Să presupunem că x_k este punctul de minim obținut cu metoda rețelei. Dacă funcția f este unimodală, atunci rezultă că minimul adevărat se află în intervalul $[x_{k-1}, x_{k+1}]$. Intervalul care încadrează minimul se numește **interval de incertitudine**. Să notăm lungimea lui cu l . Acuratețea evaluării minimului poate fi dată de mărimea $l/(b-a)$. În cazul metodei rețelei, $l = 2h = 2(b-a)/n$.

Pentru a atinge o eroare impusă $\varepsilon = 10^{-m}$ ($m \in \mathbb{N}$), rezultă că intervalul $[a, b]$ trebuie discretizat în $n = 2 \cdot 10^m$ subintervale, deoarece:

$$\frac{2h}{b-a} \leq \varepsilon \implies \frac{2}{n} \leq \varepsilon \implies n \geq \frac{2}{\varepsilon} = 2 \cdot 10^m$$

Efortul de calcul este dat de numărul de evaluări de funcții: $n+1 = 2 \cdot 10^m + 1 \approx 2 \cdot 10^m$.

Exercițiul 2.3: Alegeți o funcție unimodală dintre funcțiile test definite în exercițiul 2.1. Pentru ea, contorizați efortul de calcul (folosiți funcția Scilab `timer`) necesar atingerii unei precizii de 10^{-1} ($n = 20$), 10^{-2} ($n = 200$), 10^{-3} , 10^{-4} . Completați un tabel de tipul:

eroare	10^{-1}	10^{-2}	10^{-3}	10^{-4}
n	20	200	2000	20000
timp [s]				

Reprezentați grafic curba timp(n) și verificați că este liniară.

2.3 Metoda Fibonacci

Efortul de căutare a extremelor poate fi redus numai dacă se cunosc mai multe date referitoare la funcție. De exemplu, dacă f este o funcție unimodală, atunci se știe că utilizarea numerelor Fibonacci minimizează efortul de obținere a optimului.

Șirul lui Fibonacci este șirul definit prin:

$$\begin{aligned} a_0 &= a_1 = 1 \\ a_n &= a_{n-1} + a_{n-2} \quad n = 2, 3, \dots \end{aligned} \tag{2.1}$$

Exercițiul 2.4: Scrieți o funcție Scilab de tipul `a = numere_fibonacci(n)` care să întoarcă un vector de dimensiune `n` conținând șirul Fibonacci între 1 și `n` (valoarea $a_0 = 1$ nu este conținută în șir).

Exercițiul 2.5: Folosind funcția de la exercițiul 2.4, enumerați primele 15 numere din șirul Fibonacci.

Exercițiul 2.6: Demonstrați¹ că șirul $\left\{\frac{a_{n-1}}{a_n}\right\}$ converge către valoarea 0.62.

2.3.1 Principiul metodei

Dacă o funcție este unimodală și $x_a < x_b$ sunt două puncte situate de aceeași parte a punctului de minim x_{\min} , cel care se găsește cel mai aproape de x_{\min} furnizează o aproximație mai bună. Sunt deci adevărate implicațiile:

$$\begin{aligned} x_a < x_b < x_{\min} &\Rightarrow f(x_b) < f(x_a), \\ x_{\min} < x_a < x_b &\Rightarrow f(x_a) < f(x_b). \end{aligned}$$

Fie $I = (c, d)$ intervalul de incertitudine al soluției numerice și fie $x_a < x_b$ două puncte din acest interval. Dacă $f(x_a) < f(x_b)$ atunci punctul de minim nu se poate găsi la dreapta lui x_b . De aceea $[x_b, d)$ se poate elimina și astfel găsim un interval de incertitudine mai mic. Dacă $f(x_a) > f(x_b)$ atunci punctul de minim nu poate fi la stânga lui x_a și deci $(c, x_a]$ se poate elimina. Dacă $f(x_a) = f(x_b)$ atunci intervalele $(c, x_a]$ și $[x_b, d)$ pot fi eliminate.

Ipoteza de unimodalitate permite micșorarea succesivă a intervalului de incertitudine, până se obține o încadrare a punctului de minim suficient de mică pentru a considera eroarea de aproximare acceptabilă.

Să considerăm intervalul de incertitudine $I_k = (c, d) = (x_{L,k}, x_{U,k})$ și, în mod arbitrar să inserăm două puncte de evaluare $x_{a,k}$ și $x_{b,k}$ (figura 2.1). La următoarea iterație, intervalul va fi I_{k+1} , plasat la dreapta sau stânga, în funcție de valorile $f(x_{a,k})$ și $f(x_{b,k})$. Din punct de vedere al numărului de evaluări de funcții, căutarea ar fi eficientă dacă unul din viitoarele puncte de evaluare $x_{a,k+1}$ și $x_{b,k+1}$ ar fi unul din punctele $x_{a,k}$ și $x_{b,k}$. De exemplu să presupunem că $I_{k+1} = I_{k+1,R} = (x_{a,k}, x_{U,k})$. Atunci, punctul $x_{b,k}$, care a fost deja evaluat, se află în interiorul noului interval de incertitudine. În consecință, vom alege $x_{a,k+1} = x_{b,k}$, fiind necesară inserarea unui singur punct nou $x_{b,k+1}$.

Cum trebuie inserate punctele de căutare? Deoarece minimul poate fi în intervalul $I_{k+1,L}$ sau $I_{k+1,R}$, ele trebuie să fie egale. Rezultă că există relația (vezi figura 2.1):

$$I_k = I_{k+1} + I_{k+2} \quad (2.2)$$

¹ Indicație: Împărțiți relația de definiție la a_{n-1} și apoi treceți la limită relația obținută pentru $n \rightarrow \infty$.

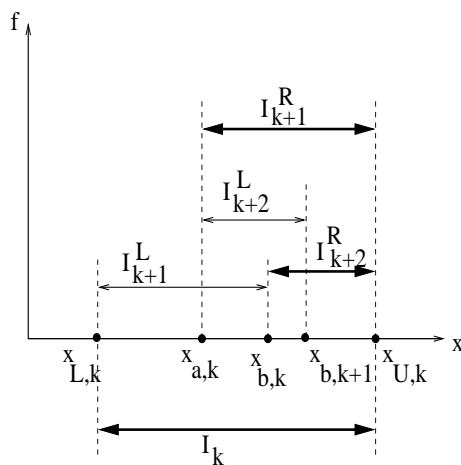


Figura 2.1: Căutarea în intervalul I_k (R = dreapta, L = stânga)

Pentru a determina lungimile intervalelor I_k vom face un raționament legat de modul în care trebuie să se termine căutarea. Ultimul interval de căutare trebuie împărțit în două și nu în trei, adică punctele x_a și x_b trebuie să coincidă. Dacă se impune o anumită lungime ultimului interval de căutare I_n (de exemplu acesta să fie o anumită fracțiune din intervalul inițial de căutare), atunci din relația (2.2) se pot determina lungimile intervalelor de căutare anterioare I_1, I_2, \dots, I_{n-1} . La final, este creat și intervalul I_{n+1} de lungime egală cu intervalul I_n , dar care nu este folosit decât pentru a determina lungimile intervalelor anterioare:

$$\begin{aligned}
 I_{n+1} &= I_n \\
 I_{n-1} &= I_n + I_{n+1} = 2I_n \\
 I_{n-2} &= I_{n-1} + I_n = 3I_n \\
 I_{n-3} &= I_{n-2} + I_{n-1} = 5I_n \\
 I_{n-4} &= I_{n-3} + I_{n-2} = 8I_n \\
 &\dots \dots \dots
 \end{aligned}$$

Șirul de coeficienți 1,1,2,3,5,8,13,21,34,55,89,144,... este șirul lui Fibonacci și de aceea căutarea bazată pe această idee se numește *căutare Fibonacci*. Dacă notăm acest șir cu a_n , se observă că există următoarea relație între lungimea intervalului final I_n și lungimea intervalului inițial I_1 :

$$\frac{I_n}{I_1} = \frac{1}{a_n} \quad (2.3)$$

După 11 evaluări de funcții reducerea intervalului de căutare este de 1/144. Se poate arăta că, dintre metodele care cer un număr fixat de evaluări de funcții, metoda Fibonacci

realizează cea mai mare micșorare a intervalului de căutare. Eficiența ei este însă mai mică decât a unei metode de ordin superior, bazate pe evaluarea derivatei.

Exercițiul 2.7: a) Arătați că $I_k/I_{k+1} = a_{n-k+1}/a_{n-k}$.

b) Exprimați I_{k+2} în funcție de I_{k+1} (această relație vă va fi utilă în înțelegerea algoritmului de mai jos).

Principalul dezavantaj al metodei Fibonacci este acela că numărul de evaluări de funcții trebuie specificat la începutul algoritmului. Dacă se cere ca valoarea funcției să scadă de un anumit număr de ori față de valoarea inițială, atunci numărul de evaluări nu poate fi estimat.

2.3.2 Algoritmul metodei

Algoritmul descris în paragraful anterior este prezentat în următorul pseudocod:

funcție [real $x_{\min}, \text{tol}_x, f_{\min}$] = met_fibonacci(real la, lb , funcție f , întreg n)

; calculează numerele Fibonacci între 1 și n

$a = \text{numere_fibonacci}(n)$

$xL(1) = la$

$xU(1) = lb$

$I(1) = lb - la$

$xa(1) = xU(1) - a(n-1)/a(n) * I(1)$

$xb(1) = xL(1) + a(n-1)/a(n) * I(1)$

$fa(1) = f(xa(1))$

$fb(1) = f(xb(1))$

$k = 1$

cât timp ($k < n - 1$)

$I(k+1) = I(k) * a(n-k)/a(n-k+1)$

dacă ($fa(k) \leq fb(k)$) **atunci**

$xL(k+1) = xL(k)$

$xU(k+1) = xb(k)$

$xa(k+1) = xU(k+1) - a(n-k-1)/a(n-k) * I(k+1)$

$xb(k+1) = xa(k)$

$fa(k+1) = f(xa(k+1))$

$fb(k+1) = fa(k)$

altfel

$xL(k+1) = xa(k)$

$xU(k+1) = xU(k)$

$xb(k+1) = xL(k+1) + a(n-k-1)/a(n-k) * I(k+1)$

```

    xa(k + 1) = xb(k)
    fa(k + 1) = fb(k)
    fb(k + 1) = f(xb(k + 1))
    k = k + 1
I(k + 1) = I(k) * a(n - k) / a(n - k + 1)
x_min = xa(n - 1)
tol_x = I(1) / a(n)
f_min = fa(n - 1)
retur

```

Exercițiul 2.8: a) Care este semnificația variabilelor folosite în algoritmul Fibonacci?

b) Transcrieți algoritmul de mai sus în Scilab.

Exercițiul 2.9: a) Verificați corectitudinea codului Scilab pentru algoritmul Fibonacci minimizând funcția $f(x) = x^2$, folosind drept prim interval de incertitudine $(-5, 15)$ și $n = 7$. Rezultatul trebuie să fie: $x_{\min} \pm \text{tol}_x = -0.238 \pm 0.95$, $f_{\min} \leq 0.0566$.

b) Pentru a observa cum s-a desfășurat căutarea, completați următorul tabel:

k	$xL(k)$	$xU(k)$	$xa(k)$	$xb(k)$	$fa(k)$	$fb(k)$	$L/R ?$
1	-5	15					
2							
3							
4							
5							
6	-1.19	0.714	-0.238	-0.238	0.0566	0.0566	—

2.3.3 Acuratețe și efort de calcul

Exercițiul 2.10: Estimați experimental dependența dintre acuratețe (știind că minimul exact este $x_{\min} = 0$, $f_{\min} = 0$) și efortul de calcul pentru minimizarea funcției de la exercițiul 2.9. Completați un tabel de tipul următor:

n	x_{\min}	tol_x	f_{\min}	timp [s]
10				
25				
50				
75				
100				

Reprezentați grafic dependența celor trei parametri ce descriu acuratețea (x_{\min} , tol_x , f_{\min}) în funcție de timpul de calcul.

2.4 Metoda secțiunii de aur

2.4.1 Principiul metodei

Plasarea primelor două puncte de căutare x_a și x_b în cazul metodei Fibonacci depinde de numărul de evaluări de funcții. Odată specificat acest număr lungimile intervalelor de incertitudine se deduc din relația (2.2) și condiția ca pentru ultimul interval de căutare punctele x_a și x_b să coincidă.

Dacă nu este pusă o astfel de condiție, trebuie găsit un alt criteriu pentru determinarea lungimii intervalelor de căutare. Vom considera și acum că relația (2.2) este satisfăcută. Un astfel de criteriu ar putea fi acela ca raportul dintre două intervale succesive de căutare să fie constant, adică

$$\frac{I_k}{I_{k+1}} = \frac{I_{k+1}}{I_{k+2}} = K. \quad (2.4)$$

Rezultă că

$$\frac{I_k}{I_{k+2}} = K^2. \quad (2.5)$$

Împărțind ecuația (2.2) cu I_{k+2} , rezultă că

$$\frac{I_k}{I_{k+2}} = \frac{I_{k+1}}{I_{k+2}} + 1. \quad (2.6)$$

Înlocuind (2.4) și (2.5) în ecuația (2.6), rezultă că

$$K^2 = K + 1, \quad (2.7)$$

ale cărei soluții sunt

$$K = \frac{1 \pm \sqrt{5}}{2}. \quad (2.8)$$

Luând numai radacina pozitivă, rezultă $K = 1.618034$.

O căutare bazată pe această schemă se numește *secțiunea de aur* deoarece împărțirea unui interval astfel încât raportul dintre cel mai mic subinterval la cel mai mare este egal cu raportul dintre cel mai mare subinterval la intervalul întreg este cunoscută sub această denumire.

2.4.2 Algoritmul metodei

Algoritmul metodei este următorul:

```

funcție [real  $x_{\min}, \text{tol}_x, f_{\min}$ ] = met_secțiunii_aur(real  $la, lb$ , funcție  $f$ , real EPS)
K_GOLDEN = ( $\sqrt{5} + 1$ )/2
 $xL(1) = la$ 
 $xU(1) = lb$ 
 $I(1) = lb - la$ 
 $xa(1) = xU(1) - I(1)/K\_GOLDEN$ 
 $xb(1) = xL(1) + I(1)/K\_GOLDEN$ 
 $fa(1) = f(xa(1))$ 
 $fb(1) = f(xb(1))$ 
 $k = 1$ 
cât timp ( $I(k) > \text{EPS}$ )
     $I(k+1) = I(k)/K\_GOLDEN$ 
    dacă ( $fa(k) \leq fb(k)$ ) atunci
         $xL(k+1) = xL(k)$ 
         $xU(k+1) = xb(k)$ 
         $xa(k+1) = xU(k+1) - I(k+1)/K\_GOLDEN$ 
         $xb(k+1) = xa(k)$ 
         $fa(k+1) = f(xa(k+1))$ 
         $fb(k+1) = fa(k)$ 
    altfel
         $xL(k+1) = xa(k)$ 
         $xU(k+1) = xU(k)$ 
         $xb(k+1) = xL(k+1) + I(k+1)/K\_GOLDEN$ 
         $xa(k+1) = xb(k)$ 
         $fa(k+1) = fb(k)$ 
         $fb(k+1) = f(xb(k+1))$ 
     $k = k + 1$ 
 $n = k$ 
 $x_{\min} = (xL(n) + xU(n))/2$ 
 $\text{tol}_x = I(1)/(K\_GOLDEN^{(n-1)})$ 
dacă ( $fa(n) \leq fb(n)$ ) atunci
     $f_{\min} = fa(n)$ 
altfel
     $f_{\min} = fb(n)$ 
retur

```

Exercițiul 2.11: a) Comparați algoritmul metodei secțiunii de aur cu algoritmul metodei Fibonacci (instrucțiuni, criteriu de oprire).

b) Implementați algoritmul metodei secțiunii de aur în Scilab.

Exercițiul 2.12: a) Verificați corectitudinea codului Scilab pentru metoda secțiunii de aur, minimizând funcția $f(x) = x^2$, folosind drept prim interval de incertitudine $(-5, 15)$ și $EPS = 1.6$. Rezultatul trebuie să fie: $x_{\min} \pm \text{tol}_x = 0.278 \pm 1.11$, $f_{\min} \leq 0.0216$.

b) Pentru a observa cum s-a desfășurat căutarea, completați următorul tabel:

k	$xL(k)$	$xU(k)$	$xa(k)$	$xb(k)$	$fa(k)$	$fb(k)$	$L/R ?$
1	-5	15					
2							
3							
4							
5							
6							
7	-0.27	0.83	0.14	0.41	0.021	0.168	—

2.4.3 Acuratețe și efort de calcul

Exercițiul 2.13: Estimați experimental dependența dintre acuratețe și efortul de calcul pentru metoda secțiunii de aur în cazul funcției $f(x) = x^2$, completând un tabel de tipul următor:

EPS	x_{\min}	tol_x	f_{\min}	n	timp [s]
1					
0.1					
0.01					
0.001					
% eps					

2.4.4 Comparație cu metoda Fibonacci

Se poate deduce o relație între numerele Fibonacci și constanta K . Pentru un număr mare n de evaluări are loc

$$a_n \approx \frac{K^{n+1}}{\sqrt{5}}. \quad (2.9)$$

Reducerea intervalului în cazul metodei Fibonacci este în consecință

$$R_F = \frac{I_n}{I_1} = \frac{1}{a_n} = \frac{\sqrt{5}}{K^{n+1}}. \quad (2.10)$$

În cazul metodei secțiunii de aur, reducerea intervalului este

$$R_{GS} = \frac{I_n}{I_1} = \frac{1}{K^{n-1}}. \quad (2.11)$$

Rezultă că

$$\frac{R_{GS}}{R_F} = \frac{K^2}{\sqrt{5}} \approx 1.17. \quad (2.12)$$

Rezultă deci că intervalul final de incertitudine în cazul metodei Fibonacci este mai mic decât în cazul metodei secțiunii de aur cu 17% (pentru același număr de evaluări de funcții). Această avantaj se obține însă pe baza specificării în avans a numărului de evaluări de funcții.

Capitolul 3

Minimizări unidimensionale - algoritmi de ordinul unu bazați pe metode de aproximare polinomială

Aceast capitol se referă la domeniul relativ simplu al minimizărilor unidimensionale. Se va presupune că printr-o metodă de decizie preliminară se poate desemna vecinătatea minimului “de interes” și că pe aceasta funcția de minimizat este unimodală (vezi figura 3.1). În capitolul anterior s-a studiat eficiența metodelor de **ordinul zero**, unde minimul era **căutat** prin generarea iterativă a unor puncte care restrângeau treptat intervalul de incertitudine. Aplicarea acestei categorii de metode (bazată doar pe evaluarea funcției nu și a derivatelor sale) conduce – mai devreme sau mai târziu – la găsirea minimului, indiferent de gradul de netezime a funcției.

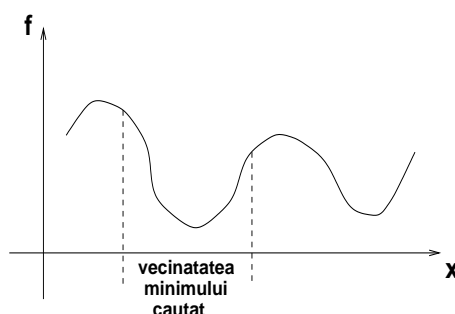


Figura 3.1: Intervalul care conține minimul căutat a fost deja determinat.

În cazul în care funcția de minimizat posedă un anumit grad de netezime, se pot aplica tehnici mai eficiente – care găsesc minimul într-un număr mai mic de pași – dar care, pe de altă parte, necesită și evaluări ale *derivatei* funcției. Tehnicile din această

categorie se bazează în general pe *aproximarea* locală a profilului funcției cu o funcție polinomială de grad mic, căreia i se poate determina ușor minimul. Algoritmii prezentați în această capitole sunt deci **de ordinul unu** (necesită evaluarea primei derivate) și aparțin clasei metodelor **de aproximare** polinomială.

3.1 Formularea problemei

Fie $f : [a, b] \rightarrow \mathbb{R}$ o funcție reală, derivabilă. Se cere să se calculeze minimul funcției f , plecând din vecinătatea acestuia.

3.2 Metoda falsei poziții (aproximării parabolice)

3.2.1 Principiul metodei

Principiul metodei constă în aproximarea funcției de minimizat, în jurul punctului de minim, cu o parabolă și alegerea minimului parabolei ca fiind o aproximare pentru minimul căutat. Procesul continuă iterativ până când derivata funcției ajunge să fie neglijabilă în minimul aproximativ.

Vom nota cu x_1, x_2, \dots, x_k șirul aproximațiilor minimului. Pentru a găsi aproximația x_{k+1} se construiește o funcție $q(x)$ pătratică (polinom de gradul doi) care să aproximeze funcția f de minimizat în jurul punctului x_k . Pentru această funcție q se pune condiția ca valorile: $q(x_k)$, $q'(x_k)$, $q''(x_k)$ să fie egale cu cele corespunzătoare ale lui f , f' și respectiv f'' în x_k . Se poate scrie:

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2. \quad (3.1)$$

Exercițiul 3.1: Cum a fost dedusă relația de mai sus ?

Exercițiul 3.2: De ce a fost impusă condiția de egalitate pentru trei valori? Dați exemple de alte seturi de condiții similare (care să asigure aceeași proprietate pentru q).

Bazându-ne pe această aproximare locală a funcției f , se poate obține o estimare “îmbunătățită” a minimului acesteia, punând condiția ca în x_{k+1} derivata întâi a lui q să se anuleze.

Exercițiul 3.3: Comentați afirmația anterioară. Cum credeți că trebuie ales x -ul inițial ?

Avem deci:

$$0 = q'(x_{k+1}) = f'(x_k) + f''(x_k)(x_{k+1} - x_k), \quad (3.2)$$

de unde:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \quad (3.3)$$

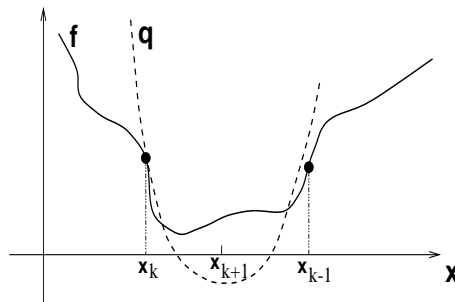


Figura 3.2: Metoda aproximării parabolice.

Relația (3.3) descrie un proces iterativ care conduce în general la obținerea rapidă a unei bune aproximații pentru minimul lui f dacă inițializarea este suficient de bună (vezi și figura 3.2). Însă, după cum se poate observa, este implicat aici și calculul derivatei a doua care, uneori, este prohibitiv din punct de vedere al resurselor necesare. Derivata a doua în x_k poate fi aproximată prin diferențe finite în funcție de derivatele întâi în două puncte, de exemplu în x_{k-1} și x_k , ceea ce conduce la

$$x_{k+1} = x_k - f'(x_k) \left[\frac{x_{k-1} - x_k}{f'(x_{k-1}) - f'(x_k)} \right]. \quad (3.4)$$

Procesul iterativ (3.4) descrie **metoda aproximării parabolice**. Acesta este un procedeu cu doi pași (x_{k+1} este calculat în funcție de x_k și x_{k-1}), de ordinul întâi (este necesară derivata de ordinul unu). În funcție de calitatea estimărilor inițiale x_1 și x_2 , iterațiile pot conduce spre minimul căutat, sau spre un punct staționar (maxim sau punct de inflexiune) nedorit. Pentru a evita această ultimă situație, algoritmul de bază ar trebui completat cu pași suplimentari de testare și corectare

Exercițiul 3.4: Cum ați concepe astfel de pași care testează soluția numerică obținută prin metoda aproximării parabolice?

Observație: Formula obținută nu implică evaluarea funcției f , deci poate fi privită și ca o metodă pentru rezolvarea problemei: $f'(x) \equiv g(x) = 0$. Metoda Newton de rezolvare a ecuațiilor neliniare adaptată pentru minimizarea funcției f conduce la o metodă de minimizare cu un pas (x_{k+1} se calculează numai în funcție de x_k), dar de ordinul doi (este necesar calculul derivatei a doua)¹.

¹În iterațiile Newton $x_{k+1} = x_k - g(x_k)/g'(x_k) = x_k - f'(x_k)/f''(x_k)$.

O altă variantă se obține prin interpolarea parabolică pe ultimii trei pași:

$$g(x) = \frac{(x - x_{k-1})(x - x_{k-2})}{(x_k - x_{k-1})(x_k - x_{k-2})}f(x_k) + \frac{(x - x_k)(x - x_{k-2})}{(x_{k-1} - x_k)(x_{k-1} - x_{k-2})}f(x_{k-1}) + \frac{(x - x_k)(x - x_{k-1})}{(x_{k-2} - x_k)(x_{k-2} - x_{k-1})}f(x_{k-2}), \quad (3.5)$$

iar soluția de la iterația $k + 1$ se obține prin rezolvarea ecuației de gradul întâi $g'(x) = 0$. Se obține astfel o metodă de minimizare numită a **interpolării parabolice**, de ordinul zero, în trei pași, care are dezavantajul că necesită o inițializare triplă (de exemplu $x_1 = a$, $x_2 = b$, $x_3 = (a + b)/2$).

3.2.2 Algoritmul metodei

Metoda aproximării parabolice de ordinul unu este prezentată în următorul pseudocod:

```
funcție [real  $x_{\min}$ ,  $f_{\min}$ , întreg  $n_{\text{iter}}$ ] = parabol_func(real  $x_1$ ,  $x_2$ ,
funcție  $f$ , funcție  $f_{\text{der}}$ , întreg NMAX, real EPS)
stopval = 1.e + 4
fdvec(1) =  $f_{\text{der}}(x_1)$ 
 $x(1) = x_1$ 
fdvec(2) =  $f_{\text{der}}(x_2)$ 
 $x(2) = x_2$ 
 $k = 2$ 
cât timp ( $k \leq \text{NMAX}$  și stopval  $\geq \text{EPS}$ )
     $x(k + 1) = x(k) - \text{fdvec}(k) * (x(k - 1) - x(k)) / (\text{fdvec}(k - 1) - \text{fdvec}(k))$ 
     $\text{fdvec}(k + 1) = f_{\text{der}}(x(k + 1))$ 
    stopval =  $|\text{fdvec}(k + 1)|$ 
     $k = k + 1$ 
 $x_{\min} = x(k)$ 
 $f_{\min} = f(x(k))$ 
 $n_{\text{iter}} = k - 2$ 
retur
```

Exercițiul 3.5: a) Care este semnificația variabilelor folosite mai sus? Care este efortul de calcul la fiecare iterație, exprimat în numărul de evaluări ale funcției sau derivatei ?

b) Cum va trebui modificat algoritmul pentru a evita memorarea tablourilor x și fdvec ?

c) Transcrieți algoritmul de mai sus în Scilab.

Exercițiul 3.6: a) Verificați corectitudinea codului scris în exercițiul anterior, considerând funcția $f(x) = \sin(x - \frac{\pi}{2})$. Folosind $x_1 = -2.0$, $x_2 = 1.0$, $\text{EPS} = 10^{-2}$ ar trebui să obțineți $n_{\text{iter}} = 3$, $f_{\text{min}} = -0.9999991$, $x_{\text{min}} = -0.0013147$. Schimbați între ele valorile inițiale și repetați testul. Comentati.

b) Repetați punctul a) pentru aceeași funcție, dar luând perechea valorilor inițiale ca fiind: -2.5 și 2.0 , sau -3.0 și 2.0 . Ce se întâmplă dacă x_1 , x_2 sunt -3.5 și 2.0 ? Comentati.

c) Observați ² cum evoluează $x(k)$ în timpul iterațiilor pentru testele efectuate la punctele a) și b).

3.2.3 Efort de calcul

Exercițiul 3.7: a) Minimizați funcția $f(x) = -\frac{\ln x}{2} + \frac{x^2}{4} + x$, folosind algoritmul aproximării parabolice și inițializarea $x_1 = 0.1$, $x_2 = 1$. Estimați experimental acuratețea și efortul de calcul al minimului (contorizând numărul de iterații și timpul de calcul), în funcție de valoarea parametrului EPS. Reprezentați grafic variația timpului de calcul și a numărului de iterații în funcție de EPS.

b) Observați ce se întâmplă în cazul $x_1 = 0.1$, $x_2 = 2$.

3.3 Metoda aproximării cubice

3.3.1 Principiul metodei

Principiul acestei metode este același cu cel al metodei aproximării parabolice, singura diferență fiind aceea că aproximarea locală a funcției de minimizat se face cu un polinom de gradul trei. Noua aproximare a minimului x_{k+1} depinde de asemenea de două puncte anterioare x_k și x_{k-1} și de valorile: $f(x_{k-1})$, $f'(x_{k-1})$, $f(x_k)$, $f'(x_k)$. Calculele corespunzătoare conduc la

$$x_{k+1} = x_k - (x_k - x_{k-1}) \left[\frac{f'(x_k) + u_2 - u_1}{f'(x_k) - f'(x_{k-1}) + 2u_2} \right], \quad (3.6)$$

unde

$$u_1 = f'(x_{k-1}) + f'(x_k) - 3 \frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}, \quad (3.7)$$

$$u_2 = \sqrt{u_1^2 - f'(x_{k-1})f'(x_k)}. \quad (3.8)$$

²Câteva sfaturi: reprezentați grafic funcția pe care o testați în intervalul (x_1, x_2) . Folosiți pentru aceasta funcția Scilab `fplot2d`. Această reprezentare grafică vă va ajuta să identificați minimul, să observați dacă funcția este unimodală, sau dacă are și alte extreme.

Exercițiul 3.8: a) Deduceți relația de mai sus.

b) Care sunt diferențele esențiale față de metoda aproximației parabolice?

3.3.2 Algoritmul metodei

Algoritmul metodei aproximării cubice de ordinul unu este prezentat în următorul pseudocod:

```
funcție [real  $x_{\min}, f_{\min}$ , întreg  $n_{\text{iter}}$ ] = cubic_func(real  $x_1, x_2$ ,
funcție  $f$ , funcție  $f_{\text{der}}$ , întreg NMAX, real EPS)

stopval = 1.e + 4
fvec(1) =  $f(x_1)$ 
fdvec(1) =  $f_{\text{der}}(x_1)$ 
 $x(1) = x_1$ 
fvec(2) =  $f(x_2)$ 
fdvec(2) =  $f_{\text{der}}(x_2)$ 
 $x(2) = x_2$ 
 $k = 2$ 
cât timp ( $k \leq \text{NMAX}$  și stopval  $\geq \text{EPS}$ )
     $u_1 = \text{fdvec}(k-1) + \text{fdvec}(k) - 3(\text{fvec}(k-1) - \text{fvec}(k))/(x(k-1) - x(k))$ 
     $u_2 = \sqrt{u_1^2 - \text{fdvec}(k-1) \cdot \text{fdvec}(k)}$ 
     $x(k+1) = x(k) - [x(k) - x(k-1)] \cdot$ 
         $(\text{fdvec}(k) + u_2 - u_1)/(\text{fdvec}(k) - \text{fdvec}(k-1) + 2u_2)$ 
    fvec( $k+1$ ) =  $f(x(k+1))$ 
    fdvec( $k+1$ ) =  $f_{\text{der}}(x(k+1))$ 
    stopval = |fdvec( $k+1$ )|
     $k = k + 1$ 
 $x_{\min} = x(k)$ 
 $f_{\min} = f(x(k))$ 
 $n_{\text{iter}} = k - 2$ 
retur
```

Exercițiul 3.9: a) Care este semnificația variabilelor folosite mai sus? Care este efortul de calcul la fiecare iterație, exprimat în funcție de numărul de evaluări ale funcției și/sau sau derivatei? Comparați cu metoda aproximării parabolice.

b) Transcrieți algoritmul de mai sus în Scilab.

3.3.3 Acuratețe și efort de calcul

Exercițiul 3.10: a) Verificați corectitudinea codului scris la exercițiul 3.9, considerând funcția $f(x) = \sin(x - \frac{\pi}{2})$. Folosind $x_1 = -2.0$, $x_2 = 1.0$ EPS = 10^{-2} ar trebui să obțineți $n_iter = 5$, $f_min = -0.9999954$, $x_min = -0.0030458$. Inversați valorile inițiale și repetați testul.

b) Pentru aceste situații, comparați eficiența și acuratețea cu cea a algoritmului aproximării parabolice. Care algoritm este mai rapid convergent (din următoarele puncte de vedere: număr de iterații, număr de evaluări de funcții, timp de calcul)?

Capitolul 4

Aplicație la minimizările unidimensionale. Determinarea indicelui de calitate al unui magnet permanent

Aceast capitol urmărește aplicarea metodelor de optimizare învățate până acum pentru dimensionarea unui magnet permanent, astfel încât indicele său de calitate să fie maxim¹.

4.1 Descrierea problemei de optimizare

Magneții permanenți sunt în general utilizați pentru producerea unei inducții magnetice într-un întrefier de dimensiuni date.

Să considerăm magnetul permanent de forma celui din figura 4.2. Să presupunem că dimensiunile întrefierului (arie și lungime) sunt date și se dorește obținerea unei anumite valori a inducției magnetice B_e cunoscute în întrefier. Să se determine dimensiunile geometrice și tipul materialului magnetului astfel încât costul magnetului să fie minim.

¹Temă pentru studenți – Concepeți un referat scris (se recomandă în Latex) care să reflecte modul în care s-au comportat metodele de optimizare studiate până acum în rezolvarea acestei probleme. Vă recomandăm ca acest referat să fie bazat pe răspunsurile la exercițiile și întrebările din acest capitol.

Fiecare student își va alege din figura 4.1 o curbă pentru care va determina indicele de calitate, așa cum se specifică în subparagrafele următoare.

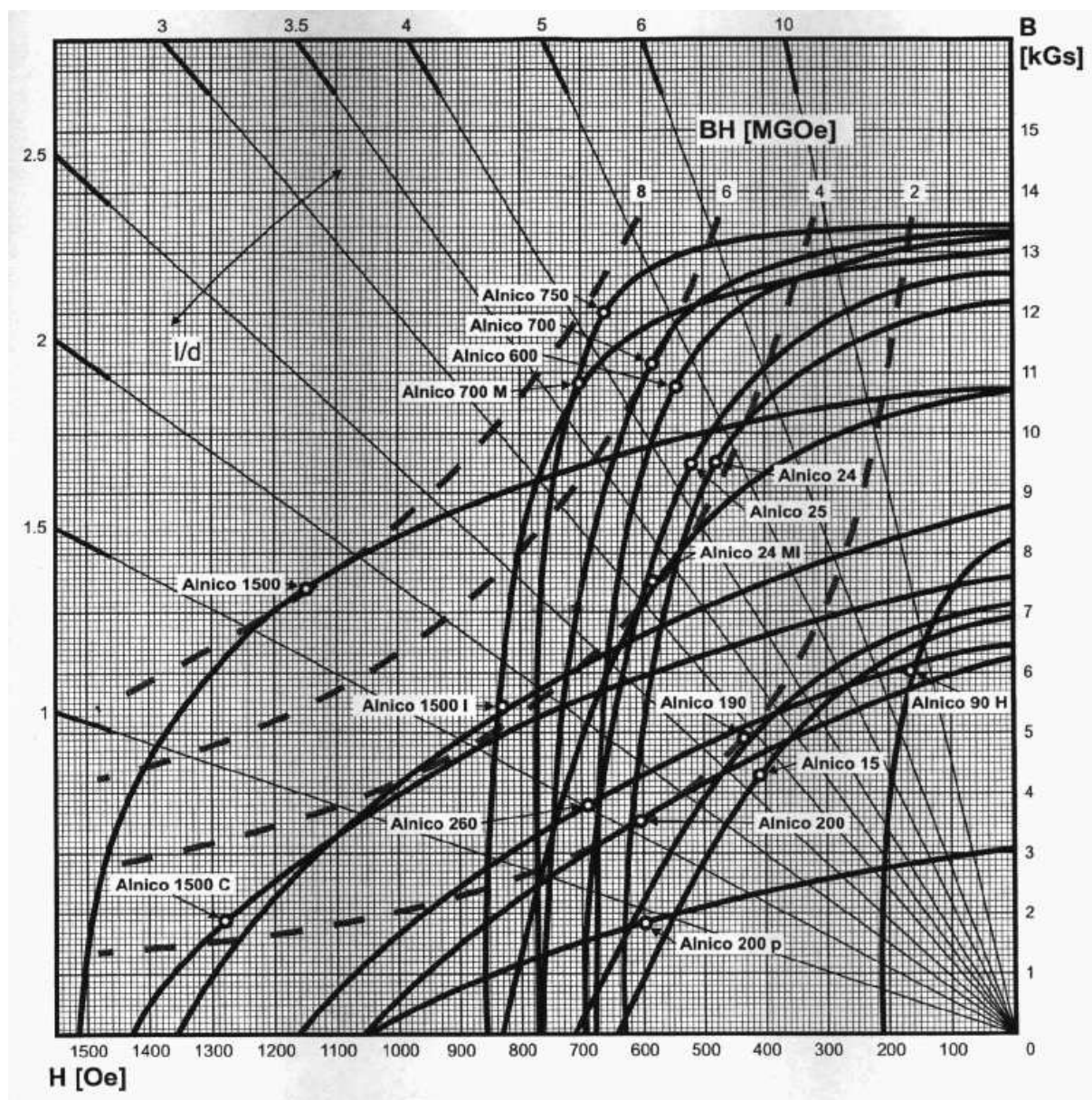


Figura 4.1: Caracteristici de material pentru materiale magnetic dure

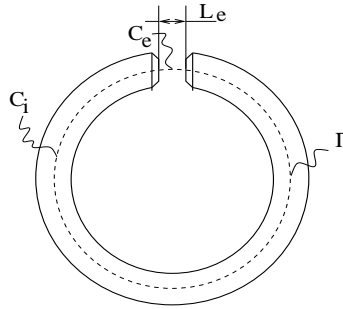


Figura 4.2: Magnet permanent

Tipul magnetului se va alege unul din cele descrise de curbele din figura 4.1 (Observație: $1 \text{ T} = 10^4 \text{ Gs}$, $1 \text{ A/m} = 4\pi 10^{-3} \text{ Oe.}$)².

4.2 Formularea problemei de optimizare

Pentru a putea formula problema ca o problemă de optimizare, trebuie făcute mai întâi câteva raționamente legate de câmpul magnetic.

Vom presupune mai întâi că modulul câmpului magnetic este constant în interiorul magnetului (valoare notată B_i) iar în întrefier vom presupune câmpul magnetic ca fiind uniform (de modul B_e). Restul câmpului de dispersie îl vom neglija.

Din legea fluxului magnetic rezultă că

$$B_e A_e = B_i A_i. \quad (4.1)$$

Exercițiul 4.1: a) Ce reprezintă A_e , A_i în formula (4.1)?

b) Indicați suprafața pe care a fost aplicată legea fluxului magnetic.

Din teorema lui Ampère rezultă că

$$H_e L_e = -H_i L_i. \quad (4.2)$$

Exercițiul 4.2: a) Ce reprezintă L_e , L_i în formula (4.2)?

b) Ce reprezintă H_e , H_i în formula (4.2)?

c) Explicați cum a fost dedusă relația (4.2).

²Valoarea citită în diviziuni pe scara inducției împărțită la 10 dă valoarea în T, iar valoarea citită în diviziuni pe scara intensității împărțită la 12.5 dă valoarea în kA/m.

Vom nota cu v_i și v_e volumele materialului magnetic și întrefierului. Rezultă atunci că:

$$v_i = L_i A_i, \quad (4.3)$$

$$v_e = L_e A_e. \quad (4.4)$$

Exercițiul 4.3: *Ce ipoteze se fac pentru a putea scrie relațiile (4.3) și (4.4)?*

Folosind relațiile de mai sus, rezultă

$$\frac{v_i}{v_e} = -\frac{B_e^2}{\mu_0 B_i H_i}. \quad (4.5)$$

Exercițiul 4.4: *Explicați cum a fost dedusă relația (4.5).*

Din relația (4.5) rezultă că, la valori date v_e și B_e în întrefier, volumul v_i al materialului magnetic (care este direct proporțional cu costul magnetului) este cu atât mai mic cu cât produsul $B_i H_i$ este mai mare. Valoarea maximă a produsului $B_i H_i = (BH)_{\max}$ măsurat în J/m³ este o mărime ce caracterizează materialele magnetice dure și se numește *indice de calitate al magnetului*.

Problema de optimizare se reduce astfel la alegerea unui punct pe caracteristica de material B-H din cadranul doi care să aibă un produs BH cât mai mare.

Exercițiul 4.5: *Explicați de ce magnetii permanenți “funcționează în cadranul doi”? Dintre materialele descrise în figura 4.1, care este materialul cu indice de calitate cel mai mare?*

Observație:

În figura 4.1 sunt desenate cu linie întreruptă curbele de BH constant. Punctele marcate cu cerculeț pe fiecare curba B-H sunt punctele în care produsul BH este maxim. De exemplu materialul Alnico 1500 are indicele de calitate 8 MGsOe.

4.2.1 Aproximarea curbei de material cu ajutorul unei expresii analitice

Ecuția curbei $B(H)$ din cadranul doi, a materialului din care se confecționează magnetii permanenți se poate aproxima cu

$$B = \frac{H + H_c}{\frac{H}{B_s} + \frac{H_c}{B_r}}, \quad (4.6)$$

în care H_c este intensitatea câmpului magnetic coercitiv, B_r este inducția magnetică remanentă și B_s este inducția magnetică de saturație.

Exercițiul 4.6: Determinați B_s astfel încât curba de magnetizare aleasă din figura 4.1 și cea dată de relația (4.6) să aibă un punct comun pentru $B = B_r/2$.

Exercițiul 4.7: a) Determinați analitic maximul expresiei BH . Fie B_m și H_m coordonatele de pe curba $B-H$ coordonatele de pe curba $B-H$ pentru care se asigură acest maxim. Observați ce relație există între B_m/H_m și B_r/H_c . Care este interpretarea grafică a punctului (B_m, H_m) ?

b) Reprezentați grafic $BH = f(H)$ pentru $H \in [-H_c, 0]$.

c) Aplicați funcției f toate metodele de optimizare învățate: Fibonacci, secțiunea de aur, metoda aproximării parabolice, metoda aproximării cubice. Analizați acuratețea și efortul de calcul.

4.2.2 Aproximarea liniară pe porțiuni a curbei de material

Exercițiul 4.8: Determinați indicele de calitate al unui material folosind o aproximare liniară pe porțiuni a curbei $B(H)$ construită astfel încât să se abată cât mai puțin de la curba din figura 4.1. Comparați rezultatele obținute cu cele din cazul anterior, în care curba de material a fost aproximată cu o relație analitică. Comparați între ele valorile B_m/H_m și B_r/H_c .

Exercițiul 4.9: Comparați rezultatele cu valorile indicate în figura 4.1.

Exercițiul 4.10: Aplicație numerică: $A_e = 1 \text{ cm}^2$, $L_e = 5 \text{ mm}$, $B_e = 1 \text{ T}$. Să se determine A_i și L_i astfel încât volumul magnetului să fie minim.

4.3 Proiectarea circuitelor magnetice cu magneți permanenți

Deoarece materialele din care se confecționează magneții permanenți sunt scumpe, circuitele magnetice conțin numai o mică porțiune de magnet permanent, jugurile fiind confecționate dintr-un material feromagnetic moale.

De exemplu, figura 4.3 reprezintă circuitul magnetic al unui instrument de măsură magnetoelectric cu magnet permanent exterior. Sistemul fix este format dintr-un magnet permanent 1 prevăzut cu piesele polare 2 și miezul cilindric 3. Bobina mobilă (nedesenată, de care este fixat acul indicator) înconjoară miezul 3, putându-se roti în întrefierul cilindric dintre piesele polare și miez. Scopul întrefierului cilindric este acela de a permite creerea unui flux magnetic cu o distribuție uniform radială, inducția magnetică păstrând o valoare constantă (de exemplu 0.3 T), independentă de unghiul de poziție al bobinei mobile.

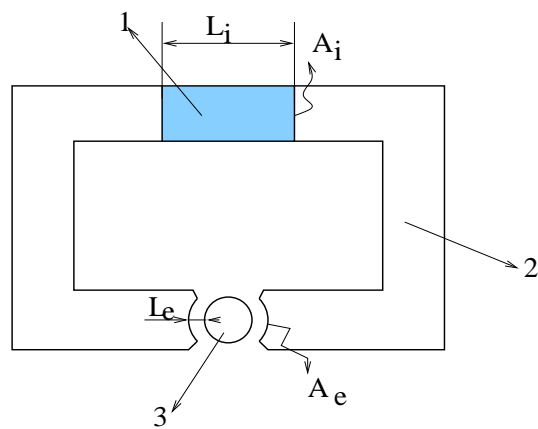


Figura 4.3: Circuitul magnetic al unui instrument de măsură magnetoelectric

Exercițiul 4.11: Aproximând în piesele polare $\mu \approx \infty$, deduceți relațiile de calcul pentru dimensiunile magnetului astfel încât volumul lui să fie minim.

Capitolul 5

Minimizări multidimensionale - metode deterministe de ordinul zero. Metoda simplexului descendent.

Metodele de optimizare pot fi clasificate în două mari categorii: metode deterministe și metode stocastice.

- **Metodele deterministe** conduc la aceeași soluție pentru rulări diferite ale programului dacă pornesc de la aceleași condiții inițiale și au aceleași parametri.
- **Metodele stocastice** au un caracter aleator și ele nu conduc în mod necesar la aceeași soluție, chiar dacă algoritmul pornește din aceleași condiții inițiale și are aceleași parametri.

Dezavantajul metodelor deterministe este acela că ele găsesc întotdeauna un extrem local, dependent de inițializare. De cele mai multe ori se dorește însă găsirea unui extrem global, lucru ce pretinde explorarea întregului domeniu de căutare și nu numai o vecinătate a inițializării. Pentru a determina un extrem global se poate proceda astfel: se execută algoritmul determinist pentru mai multe puncte inițiale de căutare împrăștiate uniform în domeniul de căutare și apoi se alege dintre soluțiile găsite valoarea cea mai bună sau se perturbă un extrem local găsit pentru a vedea dacă algoritmul determinist cu această inițializare regăsește același extrem. Ca o alternativă la aceste două abordări, au început să fie folosiți tot mai des algoritmi stocastici. Aceștia nu garantează găsirea unui extrem global, dar ei au o probabilitate mult mai mare de a găsi un astfel de extrem. De asemenea, ei mai au avantajul că nu necesită evaluarea derivatelor funcției de optimizat, fiind în consecință algoritmi de ordinul zero. Dezavantajul lor este acela că numărul de evaluări de funcții necesar pentru găsirea optimului este relativ mare față de cazul metodelor

deterministe, dar în multe situații acest sacrificiu trebuie făcut, pentru că acești algoritmi sunt singurii care dau rezultate numerice acceptabile.

În această capitol și în cel ce urmează sunt prezentați doi dintre cei mai celebri algoritmi determiniști de ordin zero (care nu au nevoie de calculul derivatelor funcției obiectiv) folosiți pentru minimizarea funcțiilor de mai multe variabile.

Prima metodă prezentată este *metoda simplexului descendent*, cunoscută și sub numele de metoda lui Nelder și Mead, după numele autorilor ei. Ea nu trebuie confundată cu “metoda simplex” de la programarea liniară¹.

O altă metodă deterministă de ordin zero folosită pentru minimizarea multidimensională este *metoda Powell*.

Deosebirea principală dintre cele două metode prezentate este aceea că metoda simplexului descendent nu are nevoie explicită de un algoritm de minimizare unidimensională ca parte a strategiei de calcul, așa cum are nevoie metoda Powell.

5.1 Formularea problemei

Fie $f : \mathbb{R}^n \rightarrow \mathbb{R}$ o funcție reală continuă. Se cere să se calculeze minimul funcției f .

Datorită complexității hiperspațiului² acesta nu poate fi explorat în totalitate și de aceea această metodă de optimizare, ca orice metodă deterministă de minimizare a funcțiilor, nu poate garanta găsirea minimului global ci doar a unuia local.

5.2 Probleme de test

Pentru testarea algoritmilor de optimizare vă recomandăm folosirea următoarelor două funcții de test cu două variabile: “cămila cu șase cocoase” și “funcția banană a lui Rosenbrock”, care sunt cunoscute ca funcții dificil de optimizat, însă simplu de evaluat.

¹Conceptul de “programare liniară” se referă la optimizarea funcțiilor liniare care au restricții liniare. Datorită faptului că funcția obiectiv este liniară, soluția problemei se află întotdeauna pe frontiera domeniului delimitat de restricții. Termenul de “programare” este sinonim în acest caz cu cel de “optimizare”, folosirea lui în acest context având doar motivații istorice.

² Fie un cub cu latura unitară în spațiul 10-dimensional. Să presupunem că o metodă de căutare a minimului a stabilit că acesta nu se găsește în volumul determinat de origine și de punctele situate la $\frac{1}{2}$ de aceasta în fiecare dimensiune. Volumul eliminat din spațiul de căutare este egal cu $(0.5)^{10} \approx \frac{1}{1000}$. Volumul în care rămâne de căutat minimul va fi 99.9% din volumul inițial, deci orice căutare exhaustivă este exclusă.

5.2.1 Cămila cu șase cocoșe

Funcția "six-hump camel back" (cămila cu șase cocoșe) are expresia

$$C(x, y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + (-4 + 4y^2)y^2, \quad (5.1)$$

unde $-3 \leq x \leq 3$ și $-2 \leq y \leq 2$. Funcția are un minim global egal cu -1.03163 în două puncte diferite: $(x, y) = (-0.0898, -0.7126)$ și $(x, y) = (0.0898, -0.7126)$. Funcția nu este deci unimodală pe acest domeniu de definiție.

Pentru ca extremul global să fie unic, funcția folosită în programul de optimizare este definită pentru $0 \leq x \leq 3$. Figura 5.1 prezintă 25 curbe de echivalori în zona $[0, 2] \times [-1.2, 1]$.

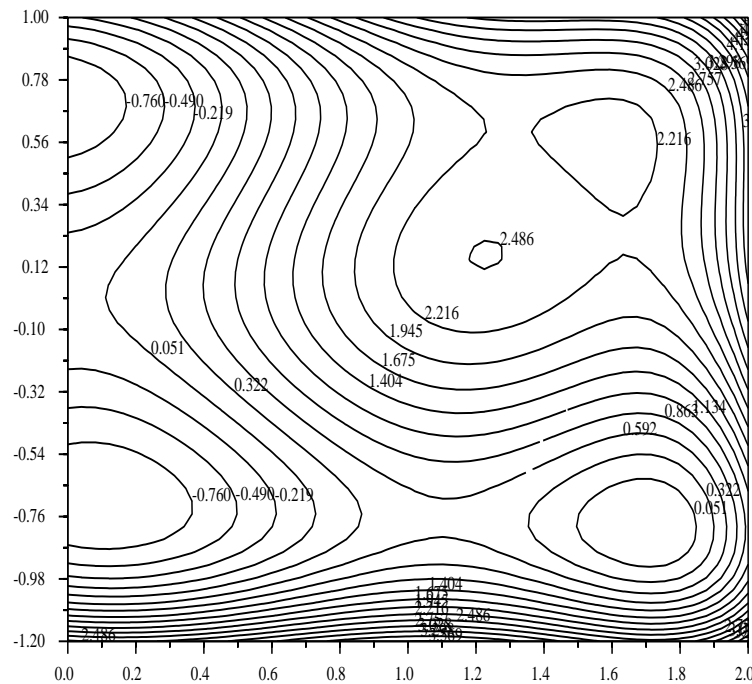


Figura 5.1: Harta funcției $C(x, y)$.

Exercițiul 5.1: a) Observați poziția și tipul extremelor din figura 5.1.

b) Implementați în Scilab funcția "cămila".

c) Scrieți un program Scilab care să permită trasarea curbelor de nivel ale acestei funcții. (Indicație: folosiți funcția `fcontour`.)

d) Scrieți un program Scilab care să permită trasarea graficului acestei funcții. (Indicație: folosiți funcția `fplot3D`.)

5.2.2 Funcția lui Rosenbrock (“banana”)

Funcția lui Rosenbrock cunoscută și sub numele de “funcția banană” are expresia

$$B(x, y) = 100(y - x^2)^2 + (1 - x)^2, \quad (5.2)$$

unde $-2.048 \leq x \leq 2.048$ și $-2.048 \leq y \leq 2.048$.

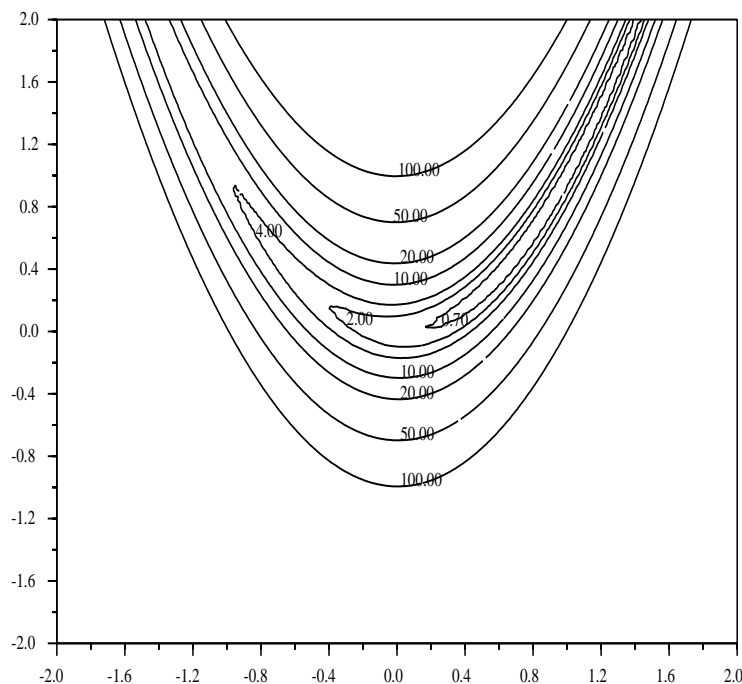


Figura 5.2: Harta funcției $B(x, y)$.

Funcția are un minim global egal cu 0 în punctul $(x, y) = (1, 1)$. Minime locale nu există, dar funcția are un relief complicat pentru algoritmi de optimizare de ordinul unu. Pentru puncte de start depărtate de minim, direcția gradientului nu indică direcția corectă către minim, valea funcției fiind răsucită.

Figura 5.2 prezintă curbele de echivalori 0.7, 2, 4, 10, 20, 50 și 100 în domeniul $[-2, 2] \times [-2, 2]$. Pentru o astfel de funcție, minimizarea la un moment dat după o singură direcție ar putea de fapt să îndepărteze punctul curent de minimul adevărat.

Exercițiul 5.2: a) *Observați poziția și tipul extremelor din figura 5.2.*

b) *Implementați în Scilab funcția “banana”.*

c) *Scrieți un program Scilab care să permită trasarea curbelor de nivel ale acestei funcții.*

d) *Scrieți un program Scilab care să permită trasarea graficului acestei funcții.*

5.3 Metoda simplexului descendent

Metoda simplexului descendent³ (datorată lui Nelder și Mead, 1965) este cumva aparte de alte metode deterministe de minimizare deoarece ea nu are nevoie explicită de algoritmi de optimizare unidimensională ca parte a strategiei de calcul. Ea are nevoie doar de evaluări de funcții nu și de derivate, fiind în consecință o metodă de ordinul zero. Nu este o metodă foarte eficientă din punct de vedere al efortului de calcul măsurat în număr de evaluări de funcții. Metoda Powell este aproape sigur mai rapidă în majoritatea aplicațiilor. Cu toate acestea metoda simplexului descendent, datorită simplității algoritmului, este utilizată frecvent pentru probleme la care efortul necesar evaluării funcției este mic. De asemenea, ea are o simplitate conceptuală și o natură geometrică care o fac “simpatică”.

5.3.1 Principiul metodei

Un **simplex** este o figură geometrică care în n dimensiuni, este un poliedru cu $n + 1$ vârfuri care conține și toate interconexiunile de tip segmente între aceste vârfuri, fețe poligonale, etc. În două dimensiuni un simplex este un triunghi, în trei dimensiuni este un tetraedru, nu neapărat regulat⁴. În general suntem interesați de simplexuri care nu sunt degenerate, adică de cele care delimitează un volum interior n -dimensional nenul.

În minimizarea unidimensională era posibil să se încadreze minimul astfel încât succesul unor izolări succesive era garantat. Nu există procedură analogă în spațiul multidimensional. În cazul multidimensional ne putem imagina că pornim cu o valoare de start oarecare (un vector cu n variabile independente) și că algoritmul își croiește drum în jos prin complexitatea de neimaginat al unei topografii n -dimensionale până când întâlnește un minim (cel puțin local).

³Downhill Simplex

⁴Metoda simplex a programării liniare nu are nici o legătură cu metoda simplexului descendent!

Metoda simplexului descendent are nevoie nu de un punct de start ci de $n + 1$ puncte de start care definesc simplexul inițial. Metoda face o serie de pași, marea majoritate doar mutând vârful simplexului căruia îi corespunde cea mai proastă valoare a funcției obiectiv (cea mai mare, în problemele de minimizare). Mutarea acestui punct se face prin fața opusă a simplexului, către o valoare mai bună. Astfel de pași se numesc **reflectări** și ei sunt construiți astfel încât să conserve volumul simplexului (și în consecință să îl păstreze nedegenerat). Pentru a accelera căutarea, simplexul poate face o **expansiune**. Când ajunge într-o vale simplexul se **contractă** în direcția transversală (se spune că se contractă parțial) și încearcă să pătrundă (să se “scurgă”) mai adânc în vale. Se poate întâmpla să existe situația în care simplexul să încerce să treacă printr-o ureche de ac și atunci el se contractă în toate direcțiile, orientându-se după punctul cel mai bun. În această situație se spune că are loc o contractare totală.

Reflectarea, expansiunea, contractarea parțială conduc la modificarea coordonatelor unui singur punct iar contractarea totală duce la modificarea coordonatelor a n puncte.

Mișcările de bază în cazul bidimensional se pot observa în figura 5.3. În această figură se presupune că P_3 este punctul în care valoarea funcției este cea mai proastă, P_1 este punctul în care valoarea funcției este cea mai bună, iar în P_2 funcția are o valoare intermediară. Cu G a fost notat mijlocul segmentului P_1P_2 iar cu indicii ' (sau '' în cazul expansiunii) punctele noi. Relația $P'_3 = G + a(G - P'_3)$ de la reflectare trebuie înțeleasă ca: $\mathbf{r}_{P'_3} = \mathbf{r}_G + a(\mathbf{r}_G - \mathbf{r}_{P'_3})$ unde \mathbf{r}_P reprezintă raza vectorială a punctului P iar a reprezintă o mărime scalară.

Mărimile a , b , g sunt mărimi scalare care se numesc coeficienți de reflectare, contractare și, respectiv, expansiune. Mărimi uzuale sunt $a = 1$, $b = 0.5$, $g = 2$.

5.3.2 Algoritmul metodei

Să notăm valorile funcției în cele trei puncte ale simplexului cu⁵ $Y_b = f(P_1)$, $Y_r = f(P_2)$ respectiv $Y_{fr} = f(P_3)$ și valoarea funcției în punctul P'_3 cu Y_{nou} . Presupunând că problema este una de minimizare, atunci Y_{nou} se poate situa într-unul din următoarele patru intervale: $(-\infty, Y_b]$, $(Y_b, Y_r]$, $(Y_r, Y_{fr}]$, (Y_{fr}, ∞) .

Un ciclu al algoritmului presupune reflectarea punctului P_3 în care funcția are valoarea cea mai mare față de mijlocul segmentului determinat de celelalte două puncte. În funcție de valoarea Y_{nou} în punctul reflectat P'_3 acesta înlocuiește punctul inițial sau nu.

Dacă valoarea funcției în punctul reflectat (Y_{nou}) este mai mică decât cea mai mică valoare a funcției în celelalte vârfuri ale simplexului (Y_b), atunci se poate încerca o expan-

⁵b = bun, r = rău, fr = foarte rău

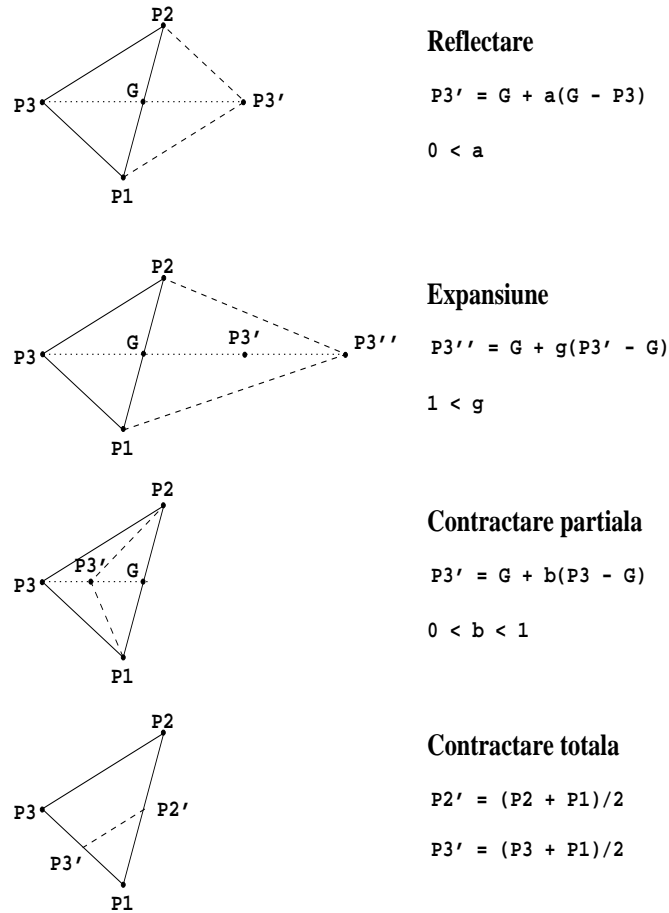


Figura 5.3: Transformări geometrice posibile.

siune în direcția punctului reflectat (s-ar putea ca aceasta să fie direcția în care se găsește minimul funcției).

Dacă Y_{nou} este, dimpotrivă, mai mare decât cea mai mare valoare a funcției în celelalte vârfuri ale simplexului (Y_{fr}), atunci direcția este greșită și trebuie contractat întregul simplex în jurul punctului în care funcția are valoare minimă (Y_b).

Dacă Y_{nou} este în intervalul (Y_b, Y_{fr}) atunci punctul reflectat înlocuiește punctul inițial. În plus, dacă valoarea funcției în punctul reflectat este în intervalul $[Y_r, Y_{fr}]$ atunci simplexul suferă o contracție (parțială sau totală).

Algoritmul descris mai sus este prezentat în următorul pseudocod:

1. Calculează vârfurile simplexului inițial P_1, P_2, P_3

repetă

2. Calculează valorile funcției în vârfurile simplexului

și ordonează punctele, $Y_b = f(P_1)$, $Y_r = f(P_2)$, $Y_{fr} = f(P_3)$

3. Calculează mijlocul segmentului P_1P_2 , față de care se va reflecta punctul cel mai rău (P_3)

4. Calculează punctul reflectat, P_3'

5. Evaluează funcția în punctul reflectat, Y_{nou}

6. **dacă** $Y_{nou} < Y_b$ **atunci** ; încerc expansiune

6.1. Calculează punctul de expansiune, P_3''

6.2. Evaluează funcția în noul punct, Y_{tmp}

6.3. **dacă** $Y_{tmp} < Y_{nou}$ **atunci** ; expansiune reușită

6.3.1. Înlocuiește P_3 cu P_3''

6.4. **altfel**

6.4.1. Înlocuiește P_3 cu P_3'

7. **altfel**

7.1. **dacă** $Y_{nou} < Y_r$ **atunci**

7.1.1. Înlocuiește P_3 cu P_3'

7.2. **altfel**

7.2.1. **dacă** $Y_{nou} < Y_{fr}$ **atunci**

7.2.1.1. Înlocuiește P_3 cu P_3'

7.2.2. Calculează punctul de contracție, P_3'''

7.2.3. Evaluează funcția în noul punct, Y_{tmp}

7.2.4. **dacă** $Y_{tmp} < Y_{fr}$ **atunci**

7.2.4.1. Înlocuiește P_3 cu P_3'''

7.2.5. **altfel** ; contracție totală

7.2.5.1. Înlocuiește P_3, P_2 cu $(P_3 + P_1)/2$, respectiv $(P_2 + P_1)/2$

până când este îndeplinită condiția de oprire

Criteriul de oprire este delicat în orice minimizare multidimensională. În mai multe dimensiuni nu există posibilitatea aplicării tehnicii de încadrare, deci nu se poate cere o anumită toleranță pentru fiecare variabilă independentă. Algoritmul poate fi oprit atunci când vectorul distanță de la un anumit pas este mai mic decât o valoare impusă. O altă posibilitate este de opri algoritmul atunci când descreșterea relativă a valorii funcției este mai mică decât o toleranță impusă. Oricare din cele două criterii de oprire pot duce la o soluție proastă datorită unui ultim pas prost făcut. De aceea se recomandă ca programele de minimizare multidimensionale să fie restartate din punctul în care se pretinde că s-a găsit minimul.

5.3.3 Exerciții

Exercițiul 5.3: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere corespunzătoare metodei simplexului descendent. Aceste fișiere constituie imple-

mentarea Scilab a metodei Nelder-Mead aplicată la determinarea minimului funcției ”cămilă”.

b) Studiați și descrieți conținutul acestor fișiere.

Observație: Calculul vârfurilor simplexului inițial se face în funcție de un singur punct P_0 și un parametru λ , astfel:

$$\mathbf{r}_{P_k} = \mathbf{r}_{P_0} + \lambda \mathbf{e}_k \quad k = 1, \dots, n, \quad (5.3)$$

unde \mathbf{e}_k reprezintă versorii axelor de coordonate.

Exercițiul 5.4: Comentați convergența algoritmului în următoarele cazuri:

- a) $P_0 = [2; 1], \lambda = 0.25$,
- b) $P_0 = [2; 1], \lambda = 0.5$,
- c) $P_0 = [2; 1], \lambda = 0.75$,
- d) $P_0 = [2; 1], \lambda = 1$,
- e) $P_0 = [0; 0], \lambda = 0.5$,
- f) $P_0 = [0; 0], \lambda = 1$,
- g) $P_0 = [0; 0], \lambda = 2$,
- h) $P_0 = [-2; 2], \lambda = 0.25$,
- i) $P_0 = [-2; 2], \lambda = 0.5$,
- j) $P_0 = [-2; 2], \lambda = 1$.

Exercițiul 5.5: Pentru una din combinațiile P_0, λ pentru care algoritmul Nelder-Mead nu a condus la rezultatul corect faceți restart din punctul declarat ca minim și observați ce se întâmplă.

Exercițiul 5.6: a) Pentru una din combinațiile P_0, λ care a condus la rezultatul corect, studiați influența parametrilor a, b și g .

c) Pentru aceeași combinație de la punctul a), studiați influența condiției de oprire.

Exercițiul 5.7: a) Modificați programul principal astfel încât să se minimizeze funcția lui Rosenbrock.

b) Studiați influența parametrului λ asupra rezultatului.

c) Evaluați efortul de calcul și acuratețea rezultatului în funcție de condiția de oprire.

Capitolul 6

Minimizări multidimensionale - metode deterministe de ordinul zero. Metoda Powell

În acest capitol este prezentată o metodă deterministă de ordin zero (care nu are nevoie de calculul derivatelor funcției obiectiv) folosită pentru minimizarea funcțiilor de mai multe variabile și anume *metoda Powell*.

Deosebirea principală dintre această metodă și metoda simplexului descendent, care este de asemenea o metodă deterministă de ordinul zero, constă în faptul că ea are nevoie explicită de un algoritm de minimizare unidimensională ca parte a strategiei de calcul. Algoritmii de minimizare folosiți sunt de tipul celor descriși în capitolul 2.

6.1 Formularea problemei

Fie $f : \mathbb{R}^n \rightarrow \mathbb{R}$ o funcție reală de n variabile, continuă. Se cere să se calculeze minimul funcției f , respectiv punctul $\mathbf{x}^* \in \mathbb{R}^n$ astfel încât $f(\mathbf{x}^*) \leq f(\mathbf{x})$ pentru orice $\mathbf{x} \in \mathbb{R}^n$.

6.2 Minimizarea după o direcție a funcțiilor de mai multe variabile

Să reamintim pentru început câteva noțiuni elementare de geometrie analitică.

Ecuția vectorială a unei drepte în plan, paralelă cu o direcție (vector) \mathbf{v} și trecând

prin punctul caracterizat de vectorul de poziție \mathbf{x}_0 se scrie sub forma

$$\mathbf{x} = \mathbf{x}_0 + \alpha \mathbf{v}, \quad (6.1)$$

unde $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ este vectorul de poziție al unui punct curent de pe dreaptă, iar α este o variabilă reală, care poate fi considerată drept o “coordonată” de-a lungul dreptei (figura 6.1). Se observă că sensul vectorului \mathbf{v} (de componente v_x și v_y) orientează dreapta D și, împreună cu punctul de coordonate x_0, y_0 , dă sens noțiunii de semn pentru α . (Coordonata lineică α este zero în punctul \mathbf{x}_0 .)

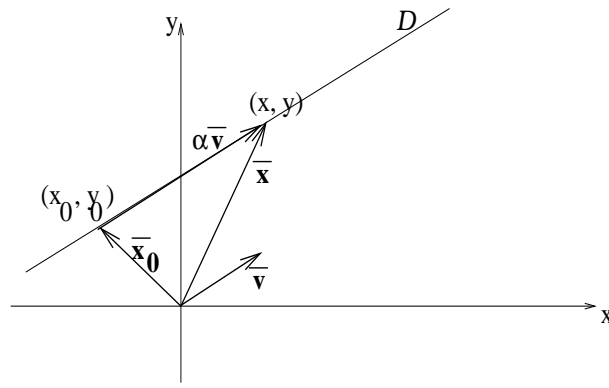


Figura 6.1: Ecuația dreptei D este $\mathbf{x} = \mathbf{x}_0 + \alpha \mathbf{v}$.

Ecuația (6.1) se transcrie identic pentru o dreaptă în spațiul n -dimensional, dar cu $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$.

Notând cu $m = v_y/v_x$ panta direcției \mathbf{v} , ecuația carteziană a dreptei care trece prin punctul (x_0, y_0) și are panta m are expresia

$$y = y_0 + m(x - x_0). \quad (6.2)$$

Exercițiul 6.1: Demonstrați (6.2) pe baza relației (6.1) scrisă pe componente.

Exercițiul 6.2: Scrieți ecuația vectorială a dreptei din spațiul tridimensional, care este paralelă la vectorul $\mathbf{v} = [1 \ 1 \ 1]^T$ și trece prin punctul $M(1, 2, 3)$. Trece această dreaptă prin origine? Ce valoare are coordonata α în punctul $P(0, 1, 2)$? Dar în punctul $Q(0, 1, 3)$?

În consecință, de-a lungul unei drepte \mathbf{v} din spațiu, funcția de n variabile

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_n) \quad (6.3)$$

poate fi scrisă ca funcție de o singură variabilă, α :

$$f(\mathbf{x}) = f(\mathbf{x}_0 + \alpha \mathbf{v}) = t(\alpha). \quad (6.4)$$

Problema minimizării funcției $f : \mathbb{R}^n \rightarrow \mathbb{R}$ pe direcția \mathbf{v} se reduce deci la problema minimizării unidimensionale a funcției $t : \mathbb{R} \rightarrow \mathbb{R}$.

Pentru a înțelege semnificația geometrică a algoritmului de minimizare după o direcție, să considerăm o funcție f de două variabile ale cărei curbe de nivel sunt cercuri. În figura 6.2 sunt reprezentate patru astfel de curbe de nivel $f(\mathbf{x}) = ct$. Punctul de minim are coordonatele $(0, 0)$, deci $f_0 > f_1 > f_2 > f_3$. Să presupunem că se dorește minimizarea funcției f după direcția \mathbf{v} , plecând din punctul inițial \mathbf{x}_0 plasat pe curba de nivel $f(\mathbf{x}) = f_0 = ct$.

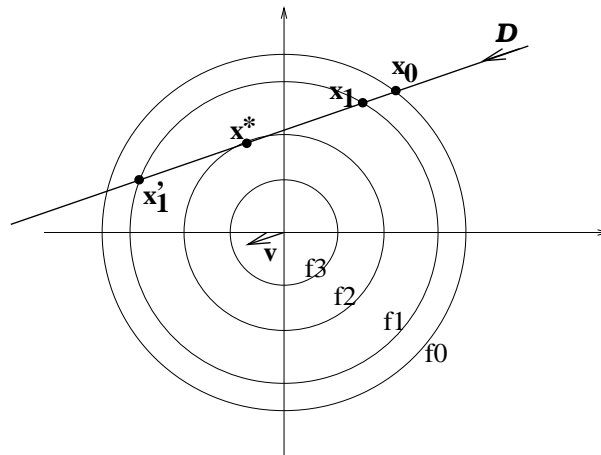


Figura 6.2: Curbe de nivel circulare.

Parcurgând dreapta D în sensul indicat pe figură, funcția f scade între punctele \mathbf{x}_0 și \mathbf{x}_1 de la valoarea f_0 la valoarea f_1 , scade în continuare între punctele \mathbf{x}_1 și \mathbf{x}^* , după care reîncepe să crească (de exemplu în punctul \mathbf{x}'_1 funcția are aceeași valoare f_1 ca și în \mathbf{x}_1 , și anume $f_1 > f_2$).

Punctul determinat ca minim pe direcția dreptei D va fi deci \mathbf{x}^* . În consecință, punctul de minim este plasat la intersecția dreptei D cu acea curbă de nivel la care D este tangentă.

Același raționament se aplică în cazul unei funcții ale cărei curbe de nivel sunt elipse, iar minimizarea se face după o dreaptă oarecare (figura 6.3).

Următorul pseudocod descrie algoritmul de minimizare pe o direcție a unei funcții de n variabile, prin căutare unidimensională. Parametrii de intrare sunt punctul \mathbf{x}_0 și direcția de căutare \mathbf{v} . Funcția întoarce coordonatele minimului pe direcția \mathbf{v} și valoarea funcției în acel punct.

funcție $[\mathbf{x}, \text{fmin}] = \text{linmin}(\mathbf{x}, \mathbf{v})$
 $[\text{alpha}, \text{tol_x}, \text{fmin}] = \text{met_sectiunii_aur}(\text{la}, \text{lb}, \text{f1dim}, \text{eps_aur});$

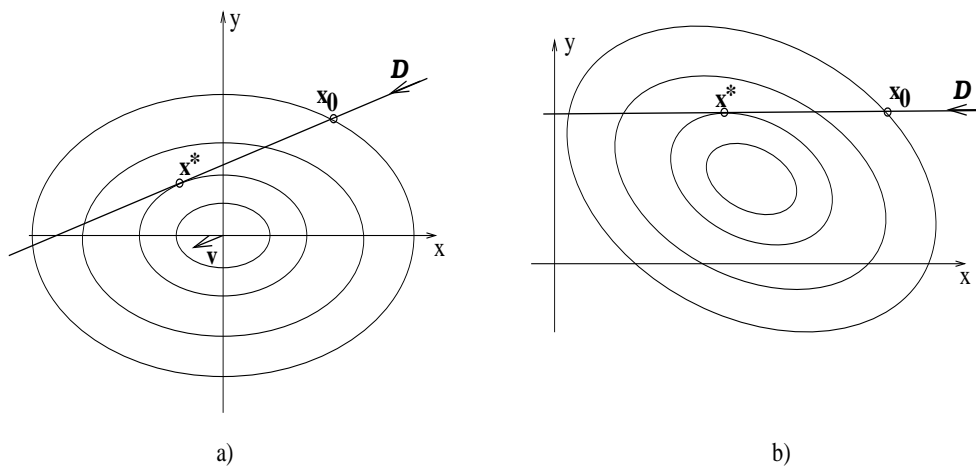


Figura 6.3: Curbe de nivel de forma unor elipse.

$x = x + \text{alpha} \cdot v;$

retur

Limitele de căutare la și lb necesare funcției `met_sectionii_aur` (vezi paragraful 2.4 de la capitolul 2) trebuie definite ca variabile globale și inițializate cu valori potrivite, în afara funcției `linmin`.

`f1dim` este funcția de o singură variabilă reală care implementează formula (6.4). Dimensiunea n a spațiului, punctul x și direcția v sunt, pentru această funcție, variabile globale: ea nu poate avea decât un singur parametru.

funcție `[t] = f1dim(alpha)`

$xt = x + \text{alpha} * v;$

$t = f(xt);$

retur

Exercițiul 6.3: Considerați funcția ale cărei curbe de nivel sunt reprezentate în figura 6.3 a) și punctul inițial x_0 . Unde va fi plasat punctul de minim după direcția unei drepte paralele cu axa Ox ?

Exercițiul 6.4: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere Scilab pentru metoda minimizării unei funcții de n variabile pe o direcție.

b) Descrieți conținutul acestor fișiere. Ce diferențe importante există între algoritmi descriși mai sus și implementarea lor în Scilab propusă în aceste fișiere?

Observație. Programul principal `main_mindir` realizează minimizarea funcției

$$f(x, y) = \frac{(x \cos \theta - y \sin \theta)^2}{4} + \frac{(x \sin \theta + y \cos \theta)^2}{400},$$

după o direcție din spațiul bidimensional.

Exercițiul 6.5: Pentru $\theta = \pi/3$ și pornind din punctul $\mathbf{x}_0(2, 2)$:

a) Minimizați funcția f după direcțiile: $\mathbf{v} = [1, 1]^T$, $\mathbf{v} = [1, 0.5]^T$ și $\mathbf{v} = [1, 2]^T$. Notați soluția, timpii de calcul și valorile minime obținute.

b) Determinați direcțiile dreptelor: (D1): $y = x \sin \theta$, (D2): $y = -x \cos \theta$ și minimizați funcția f după aceste direcții. Notați timpii de calcul și valorile minime obținute.

c) După care din direcții se obține cea mai mică valoare a funcției? Găsiți o explicație pentru aceasta.

Exercițiul 6.6: Studiați și comentați influența inițializării asupra valorii minime obținute:

a) Considerați punctele $(2, 2)$, $(2, -2)$, $(-2, 2)$, $(2, -2)$. (Observație. Pentru a ușura comparația, nu reafixați curbele de nivel la fiecare nouă inițializare.)

b) Considerați de asemenea setul de puncte $(1, 1)$, $(1, -1)$, $(-1, 1)$, $(-1, -1)$.

Exercițiul 6.7: Studiați influența scării folosite în definirea direcției \mathbf{v} .

Alegeți $\mathbf{x}_0 = (2, 2)$, direcțiile $\mathbf{v} = [1, 1]^T$ și $\mathbf{v} = [1, 2]^T$ și scalați vectorii \mathbf{v} cu coeficienții $K = 0.1, 10, 1000$. Ce observați? Care este explicația fenomenului?

6.3 Principiul metodei Powell

Minimul funcției f pe o direcție \mathbf{v} nu este, evident, minim local și cu atât mai puțin minimul global al funcției f , în întregul spațiu n -dimensional. Pentru a găsi minimul global ar trebui investigat întregul spațiu, prin căutări succesive pe n direcții liniar independente \mathbf{v}_i , $i = 1, \dots, n$, de exemplu pe direcțiile versorilor celor n axe.

Minimizarea se face astfel: pornind dintr-un punct inițial \mathbf{x}_0 , se determină minimul pe prima direcție și se notează noul punct găsit cu \mathbf{x}_1 ; pornind din \mathbf{x}_1 , se determină minimul pe a doua direcție, \mathbf{v}_2 și se notează noul punct cu \mathbf{x}_2 ; ș.a.m.d. În general,

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha \mathbf{v}_i, \quad (6.5)$$

cu α determinat prin minimizare pe direcția \mathbf{v}_i . Dacă după parcurgerea tuturor celor n direcții ale spațiului diferența valorilor funcției în punctul inițial și final, $|f(\mathbf{x}_0) - f(\mathbf{x}_n)|$ este “mare”, înseamnă că putem fi încă departe de minim, și atunci: se alege o nouă inițializare (de exemplu, se pleacă din ultimul punct găsit, $\mathbf{x}_0 = \mathbf{x}_n$), și se reia algoritmul. “Iterațiile” se opresc atunci când scăderea funcției prin parcurgerea celor n direcții devine nesemnificativă.

Exercițiul 6.8: Dați o definiție cuvântului “iterație” în contextul unei astfel de metode (de minimizare prin căutare pe un set de direcții).

În figura 6.4 este prezentat progresul iterațiilor pentru o funcție de două variabile, cu direcțiile de căutare egale cu versorii axelor: $\mathbf{v}_1 = \mathbf{i}$, $\mathbf{v}_2 = \mathbf{j}$. Se constată că, în cazul funcției din figură, caracterizată de o “vale” îngustă în care se găsește minimul, algoritmul avansează spre minim cu pași mici făcuți în direcțiile axelor de coordonate. Convergența este deosebit de înceată, pentru că nici una din direcțiile axelor nu se întâmplă să coincidă cu axa “văii”.

De aceea se caută tehnici de alegere a unui set de direcții de căutare mai bune, plasate în lungul văii, care să conducă la un avans mai rapid către minim. În plus, dacă direcțiile nu sunt ortogonale, este de dorit ca ele să fie astfel alese încât minimizarea pe o direcție să nu compenseze efectul minimizărilor efectuate anterior pe alte direcții, ceea ce ar conduce la proceduri neconvergente.

Algoritmii de ordinul 0 bazați pe minimizare unidimensională pornesc de la o inițializare a întregului set de direcții \mathbf{v}_i și “ajustează”, la sfârșitul fiecărei iterații, fie întreg setul de direcții, fie numai o parte din ele. Ajustarea se face numai pe baza informațiilor asupra proprietăților funcției, obținute prin minimizarea de la iterația curentă.

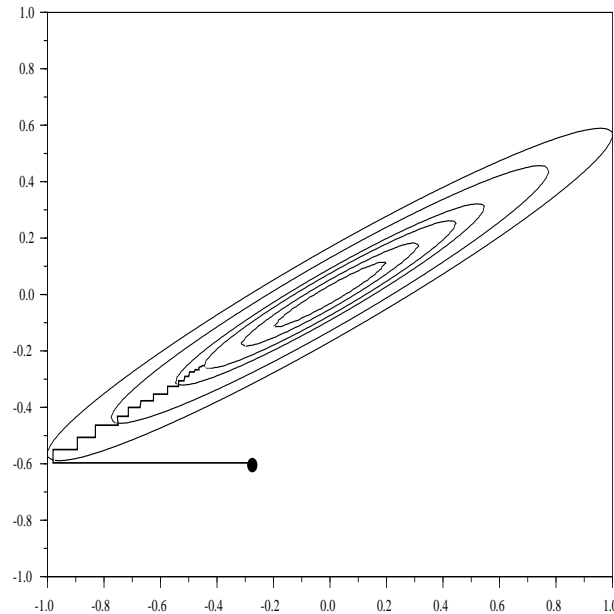


Figura 6.4: Evoluția iterațiilor la metoda de optimizare prin căutare pe un set de direcții.

În metoda Powell, direcțiile de căutare sunt ajustate după fiecare iterație, eliminând o direcție din cele n și adăugând în schimb direcția \mathbf{v}_m a “deplasării medii” a soluției pe parcursul ultimei iterații,

$$\mathbf{v}_m = \mathbf{x}_n - \mathbf{x}_0. \quad (6.6)$$

Noua inițializare se determină efectuând încă o minimizare pe direcția \mathbf{v}_m :

$$\mathbf{x}_0 = \mathbf{x}_n + \alpha \mathbf{v}_m. \quad (6.7)$$

Elementul cheie pentru viteza de convergență a algoritmului și pentru corectitudinea soluției este alegerea direcției eliminate (care urmează să fie înlocuită cu \mathbf{v}_m). Se pot folosi două tehnici:

- Se elimină întotdeauna \mathbf{v}_1 , se renumerează direcțiile (\mathbf{v}_2 devine \mathbf{v}_1 , etc.) și se adaugă \mathbf{v}_m ca ultimă direcție. Vom numi metoda astfel obținută **metoda Powell de bază**.

Această tehnică dă rezultate proaste pentru anumite tipuri de funcții, deoarece pe parcursul iterațiilor noile direcții adăugate tind să devină aproape coliniare cu unele din direcțiile existente.

Explicația este destul de subtilă și va fi ilustrată printr-un exemplu. Să notăm cu \mathbf{v}_j (în general cu $j \neq 1$), direcția după care, la iterația curentă, funcția a avut cea mai mare scădere. Este probabil ca \mathbf{v}_j să fie o componentă importantă și în deplasarea medie \mathbf{v}_m (vezi figura 6.5 a) pentru o exemplificare în cazul $n = 2$, în care cea mai mare deplasare a avut loc după direcția \mathbf{v}_2 iar \mathbf{v}_m este aproape colinar cu \mathbf{v}_2).

Cele două direcții \mathbf{v}_j și \mathbf{v}_m devin aproape coliniare și păstrarea ambelor în setul de direcții pentru iterația următoare face ca, practic, doar $n - 1$ direcții să fie liniar independente. În consecință, minimul determinat de algoritm va fi complet greșit, deoarece căutarea se face într-un subspațiu de dimensiune $n - 1$.

- Se elimină direcția cea mai bună, \mathbf{v}_j , cea după care funcția a avut cea mai mare scădere la iterația curentă, și se adaugă direcția medie \mathbf{v}_m . Vom numi metoda astfel obținută **metoda Powell modificată**.

O exemplificare este prezentată în figura 6.5. La iterația curentă k , cea mai mare scădere a avut loc după direcția $\mathbf{v}_j = \mathbf{v}_2$. Dacă, pentru iterația următoare $k + 1$ s-ar păstra direcțiile \mathbf{v}_2 și \mathbf{v}_m , acestea ar fi aproape coliniare. Algoritmul s-ar comporta atunci ca și cum minimizarea s-ar face, practic, de la iterația $k + 1$ încolo, numai după direcția axei Oy .

În algoritmul propus în cele ce urmează, pentru a economisi memorie, se vor folosi doar doi vectori pentru memorarea soluției: \mathbf{x}_{i-1} din relația (6.5) se notează cu \mathbf{xv}

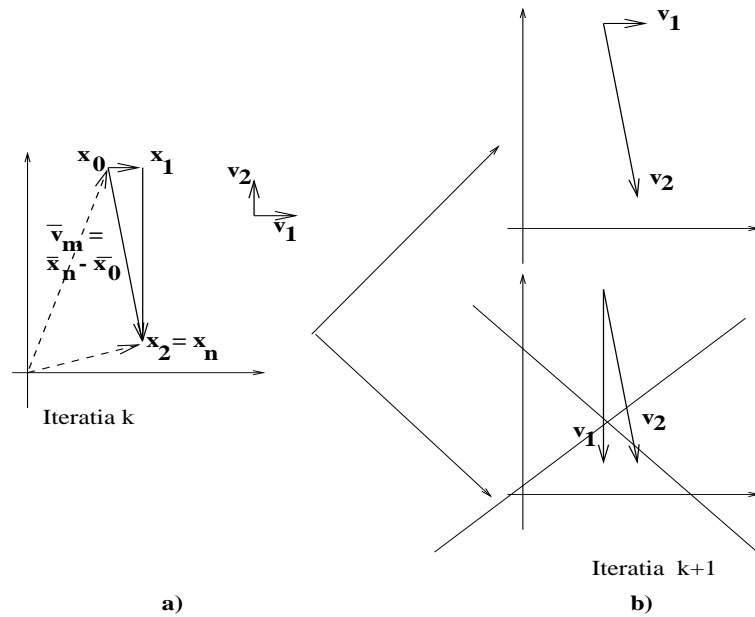


Figura 6.5: a) Direcția cea mai bună este v_2 și direcția medie de deplasare este v_m . b) Dacă se păstrează vechiul v_2 , atunci minimizarea se face numai după direcția Oy .

(“vechi”), iar x_i din aceeași relație se notează cu x_n (“nou”). În plus, este necesară memorarea inițializării x_0 în vederea calculării direcției medii.

În consecință, algoritmul metodei Powell modificate este următorul.

Date: inițializarea x_0 , direcțiile inițiale $v_i, i = 1, \dots, n$

$\text{deltaf} = 1$

$f_0 = f(x_0)$

repetă

$\text{dfmax} = 0$; cea mai mare scădere a funcției

$j = 0$; direcția cu cea mai mare scădere

$x_v = x_0, f_v = f_0$

pentru $i = 1, n$; parcurge direcțiile

$[x_n, f_n] = \text{linmin}(x_v, v_i)$; minimizează pe direcția v_i

dacă $|f_n - f_v| > \text{dfmax}$ **atunci** ; am găsit o scădere mai mare

$\text{dfmax} = |f_n - f_v|$

$j = i$

$x_v = x_n, f_v = f_n$; actualizează x și $f(x)$

$v_m = x_n - x_0$; direcția medie

```

deltaf = | fn - f0 |      ; variația funcției la iterația curentă
vj = vn                ; pune direcția vn pe poziția j
vn = vm                ; pune direcția vm pe ultima poziție
[x0,f0] = linmin(xn, vm) ; determină noua inițializare
până când deltax < eps

```

Este de notat că în algoritm se preferă adăugarea noii direcții \mathbf{v}_m pe ultima poziție în șirul de direcții (pentru aceasta, se mută întâi pe poziția j direcția aflată pe poziția n , apoi se adaugă \mathbf{v}_m pe poziția n).

În practică, este bine să se folosească o condiție de oprire de tip relativ și nu absolut, de exemplu raportând **deltax** la media valorilor funcției în punctele \mathbf{x}_0 și \mathbf{x}_n :

$$\frac{|f(\mathbf{x}_n) - f(\mathbf{x}_0)|}{1/2 (|f(\mathbf{x}_n)| + |f(\mathbf{x}_0)| + \text{EPS})} < \text{ftol}, \quad (6.8)$$

unde EPS este un estimator al erorii relative de rotunjire a mașinii iar ftol este o toleranță adimensională impusă.

6.4 Îmbunătățiri ale metodei Powell

S-a constatat că în anumite cazuri este de preferat ca setul de direcții utilizate la o iterație să se păstreze neschimbat.

Pentru a cuantifica aceste situații să mai considerăm un punct “extrapolat” plasat pe direcția \mathbf{v}_m (figura 6.6):

$$\mathbf{x}_E = 2\mathbf{x}_n - \mathbf{x}_0 = \mathbf{v}_m + \mathbf{x}_n,$$

în care funcția are valoarea $f_E = f(\mathbf{x}_E)$. Considerații simple de geometrie plană arată că punctul \mathbf{x}_E este plasat pe direcția \mathbf{v}_m și că $|\mathbf{x}_n - \mathbf{x}_0| = |\mathbf{x}_E - \mathbf{x}_n| = |\mathbf{v}_m|$.

Să mai notăm cu $f_0 = f(\mathbf{x}_0)$, $f_n = f(\mathbf{x}_n)$ și cu Δf_{\max} modulul celei mai mari scăderi a funcției (cea pe direcția \mathbf{v}_j).

Situațiile în care este de preferat ca setul de direcții să se păstreze neschimbat sunt următoarele:

1. Continuând deplasarea de-a lungul direcției \mathbf{v}_m funcția crește în loc să scadă.

Pentru a înțelege ideea, analizați cu atenție figura 6.7. Pentru funcția din figura a, minimumul este plasat între \mathbf{x}_n și \mathbf{x}_E (\mathbf{x}_E este plasat pe o curbă de nivel de valoare mai

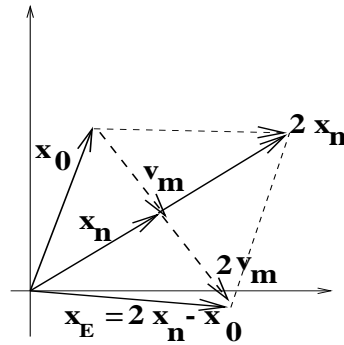


Figura 6.6: Construcția punctului extrapolat x_E .

mică decât valoarea curbei pe care e plasat x_0), deci direcția v_m merită explorată în continuare. Pentru funcția din figura b, în care x_E este plasat pe o curbă de nivel de valoare mai mare decât valoarea curbei pe care e plasat x_0 , deplasarea în continuare pe direcția v_m până în x_E poate conduce doar la o creștere a funcției, deci v_m nu este o direcție bună pentru iterația următoare.

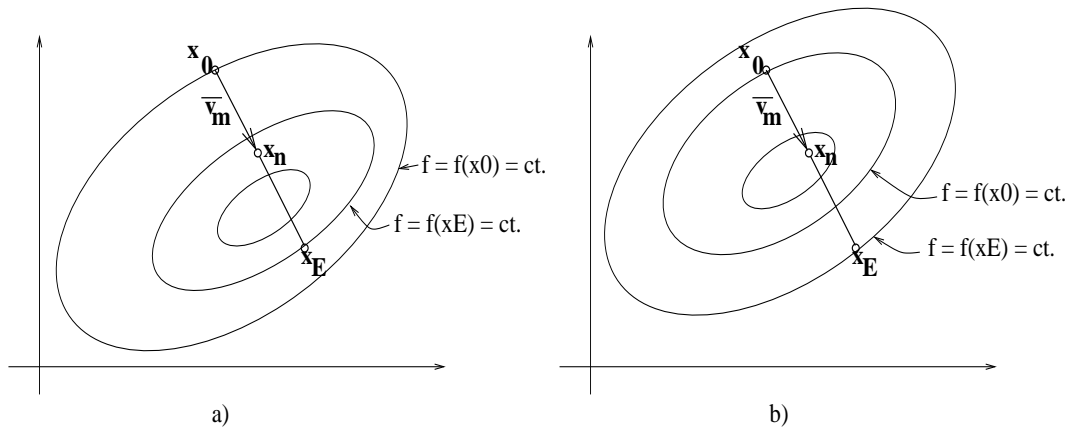


Figura 6.7: În situația b, direcția $v_m = x_n - x_0$ nu merită introdusă în setul de direcții pentru iterația următoare.

În consecință, direcțiile de căutare **nu trebuie modificate** dacă

$$f_E \geq f_0. \quad (6.9)$$

2. La iterația curentă, scăderea funcției nu se datorează cu precădere unei anumite direcții, deci merită continuarea explorării pe direcțiile existente.

Pentru a putea exprima matematic această situație, se fac următoarele raționamente.

Dacă procedura este convergentă, pe parcursul iterațiilor curente funcția a scăzut de la valoarea f_0 la valoarea f_n . În general, această scădere este mai mare decât scăderea Δf_{\max} pe cea mai bună direcție \mathbf{v}_j , (deoarece și celelalte direcții au contribuit la minimizare), deci avem $f_0 - f_n \geq \Delta f_{\max}$.

Dacă scăderea de la \mathbf{x}_0 la \mathbf{x}_n s-a datorat **cu precădere** direcției \mathbf{v}_j , atunci $f_0 - f_n$ nu diferă cu mult de Δf_{\max} (vezi comentariul referitor la eliminarea celei mai bune direcții). Dacă însă $f_0 - f_n \gg \Delta f_{\max}$, înseamnă că mai multe direcții, nu numai \mathbf{v}_j , au contribuit la scădere. Nu există deci nici o direcție “foarte bună” care să merite eliminată, ci este mai bine să fie păstrate toate cele existente. Întrebarea este cum să cuantificăm “mult mai mare”. Putem considera ca “referință” scăderea de-a lungul direcției \mathbf{v}_m , între \mathbf{x}_0 și punctul extrapolat \mathbf{x}_E , egală cu $f_0 - f_E$ și să punem o condiție de **păstrare neschimbată a setului de direcții** de forma

$$K|f_0 - f_n - \Delta f_{\max}| \geq |f_0 - f_E|,$$

sau

$$K'(f_0 - f_n - \Delta f_{\max})^2 \geq (f_0 - f_E)^2, \quad (6.10)$$

cu $K' = K^2$ un “coeficient de siguranță” supraunitar (de exemplu $K' = 2$).

3. Ultimul punct \mathbf{x}_n obținut la iterația curentă se află într-o zonă în care derivatele de ordinul al doilea au valori considerabile, deci s-ar putea ca \mathbf{x}_n să fie deja în zona minimului.

Derivata a doua a funcției în punctul \mathbf{x}_n poate fi aproximată printr-o formulă de diferențe centrate, deoarece dispunem de valorile funcției în punctele diviziunii echidistante $[\mathbf{x}_0, \mathbf{x}_n, \mathbf{x}_E]$, cu $|\mathbf{x}_n - \mathbf{x}_0| = |\mathbf{x}_E - \mathbf{x}_n| = |\mathbf{v}_m|$:

$$f''(\mathbf{x}_n) \approx \frac{f_0 - 2f_n + f_E}{|\mathbf{v}_m|^2}.$$

Modulul derivatei a doua este deci proporțional cu $f_0 - 2f_n + f_E$. Pentru a obține o valoare relativă, această expresie se împarte la o mărime de dimensiunea funcției f , de exemplu la Δf_{\max} .

În consecință, direcțiile de căutare **nu trebuie modificate** dacă

$$\frac{|f_0 - 2f_n + f_E|}{\Delta f_{\max}} \text{ are o valoare mare.} \quad (6.11)$$

Acestea sunt cele trei cazuri în care este mai bine ca direcțiile existente să fie păstrate și la iterația următoare.

În practică, deoarece nu dispunem de un termen de comparație pentru derivata a doua (nu știm ce înseamnă “derivata a doua este mare”), formulele (6.10) și (6.11) se

combină într-o singură formulă (mai jos, $K' = 2$):

$$\frac{|f_0 - 2f_n + f_E|}{\Delta f_{\max}} \cdot 2(f_0 - f_n - \Delta f_{\max})^2 \geq (f_0 - f_E)^2. \quad (6.12)$$

Rezumând, de la o iterație la alta direcțiile de căutare **nu se modifică** dacă este îndeplinită una din condițiile (6.9) sau (6.12).

6.5 Viteza de convergență

Se poate demonstra că metoda Powell de bază (la care ajustarea direcțiilor se face eliminând întotdeauna \mathbf{v}_1 și adăugând \mathbf{v}_m ca ultimă direcție) aplicată unei funcții pătratice de n variabile conduce la minim după n iterații, deci în total după $n(n+1)$ minimizări unidimensionale. Se spune că metoda are ordinul de complexitate pătratic, deoarece numărul de minimizări necesare pentru a determina minimul este (pentru o clasă precizată de funcții) de ordinul $O(n^2)$.

Observație. În cazul funcțiilor de o singură variabilă ($n = 1$), o funcție pătratică are forma

$$f(x) = ax^2 + bx + c,$$

iar în n dimensiuni are forma

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c,$$

cu \mathbf{A} o matrice de dimensiune $n \times n$, \mathbf{x} și \mathbf{b} vectori n -dimensionali și c o constantă reală. Funcțiile ale căror curbe de nivel sunt reprezentate în figurile 6.2 și 6.3 sunt pătratice.

Dacă funcția nu este pătratică, vor fi în general necesare mult mai multe iterații pentru a ajunge, cu metoda Powell de bază, în zona minimului.

În metoda Powell modificată, schimbarea strategiei de ajustare a direcțiilor conduce la pierderea ordinului de complexitate pătratic al algoritmului (el nu va mai converge în doar n iterații nici pentru o funcție pătratică), în schimb acesta devine mai robust.

Exercițiul 6.9: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere Scilab pentru metoda Powell.

b) Descrieți conținutul acestor fișiere.

c) Executați programul `main_pow.sci`. Programul minimizează o funcție de mai multe variabile prin metoda Powell îmbunătățită și afișează soluția, valoarea funcției în soluție, numărul de iterații și timpul de calcul.

De asemenea, programul reprezintă grafic evoluția soluției, unind prin linii de două culori punctele succesive.

d) Ce semnificație au cele două culori?

Exercițiul 6.10: Modificați programul `main_pow` pentru a minimiza funcția

$$f(x, y) = \frac{(x \cos \theta - y \sin \theta)^2}{4} + \frac{(x \sin \theta + y \cos \theta)^2}{400},$$

pentru diverse valori ale parametrului θ : $0, \pi/6, \pi/3, \pi/2, 2\pi/3, 3\pi/4, 5\pi/6, \pi$.

a) Notați, pentru fiecare valoare a unghiului, numărul de iterații efectuate și timpul total de calcul.

b) Pentru fiecare valoare a unghiului, determinați timpul mediu consumat într-o iterație. Se obține aceeași valoare pentru toate valorile θ ? Dacă da, de ce?

c) Pentru valorile θ din cadranul I explicați convergența diferită a procedurii pentru diverse unghiuri.

d) Explicați diferența de viteză de convergență între unghiurile plasate în cadranul întâi și simetricele lor față de axa Oy (plasate în cadranul al doilea).

Exercițiul 6.11: Pentru funcția definită la exercițiul 6.10 cu $\theta = \pi/3$, comparați cele două variante ale metodei Powell (simplă și respectiv cu îmbunătățiri).

Observație. Metoda cunoscută în literatura de specialitate sub numele de “metoda Powell” este cea îmbunătățită.

Exercițiul 6.12: Pentru funcția definită la exercițiul 6.10 cu $\theta = \pi/3$, studiați influența următorilor parametri asupra convergenței și asupra timpului de calcul:

- Limitele domeniului de integrare în metoda secțiunii de aur (la și lb);
- Eroarea impusă în metoda secțiunii de aur;
- Toleranța impusă funcției (ftol);
- Eroarea impusă în metoda secțiunii de aur;
- Estimatorul EPS al erorii relative de rotunjire;
- Inițializarea \mathbf{x}_0 .

Exercițiul 6.13:

a) Printr-o schimbare de variabilă, transformați funcția f definită la exercițiul 6.10 într-o funcție h ale cărei curbe de nivel să fie cercuri.

b) Ce legătură există între valoarea minimă a funcției f și valoarea minimă a funcției h ? Dar între punctele \mathbf{x}_f și \mathbf{x}_h în care cele două funcții își ating minimul?

c) Verificați dacă o funcție de forma funcției h există deja implementată în fișierul `functii.sci`. Dacă nu, scrieți codul pentru funcția h .

Minimizați funcția h obținută din f pentru $\theta = \pi/3$ și pornind din punctul $P = (2, 2)$ (în sistemul inițial de coordonate), cu ambele variante ale metodei Powell. Ce observați?

Notă. Transformați și coordonatele punctului inițial în noul sistem de coordonate.

Exercițiul 6.14: a) Minimizați funcția “cămilă cu șase cocoase” pornind de la următoarele inițializări: $\mathbf{x}_0 = (2, 2), (-2, 2), (-2, -2), (2, -2), (2, 1)$. Comentați rezultatele obținute.

Pentru aceeași funcție, folosiți valoarea inițială $\mathbf{x}_0 = (1, 1)$ și minimizați cu ambele variante ale algoritmului Powell. Comentați.

b) Comparați numărul de iterații, timpul de calcul și fiabilitatea (numărul de “rateuri”) cu cele de la metoda simplex (în cazul inițializărilor utilizate pentru această funcție și la exercițiul 5.4).

Comentați eficiența celor două metode în cazul funcției cămilă.

c) Comentați fiabilitatea metodei Powell în cazul funcțiilor care prezintă și minime locale pe lângă cele globale.

Exercițiul 6.15: Minimizați funcția “banană” prin algoritmul Powell.

a) Studiați influența parametrilor la și lb asupra vitezei de convergență și asupra timpului de calcul. Alegeți de exemplu $(la, lb) = (-1, 1)$ și $(-1000, 1000)$.

b) Studiați influența inițializării \mathbf{x}_0 asupra convergenței. Alegeți $\mathbf{x}_0 = (2, 2), (0, 0), (-2, 2), (2, -2), (-2, -2), (3, 3)$.

Pentru inițializările la care metoda îmbunătățită nu converge, încercați varianta simplă a metodei Powell.

Justificați eșecul minimizării în cazul punctului inițial $(3, 3)$.

Exercițiul 6.16: Minimizați funcția de trei variabile

$$f(x, y, z) = x^2 + x^2y^2 + (z - 1)^2 + 1.$$

Observația 1. Minimul este egal cu 1 și se obține în punctul $(0, 0, 1)$.

Observația 2. Evoluția soluției nu va fi afișată în cazul acestei funcții, deoarece funcțiile Scilab care fac afișări grafice pot fi apelate doar în două dimensiuni.

Capitolul 7

Minimizări multidimensionale - metode deterministe de ordinul unu. Metoda gradientilor conjugați.

Metodele deterministe de ordinul unu determină minimumul funcțiilor multidimensionale unimodale fără restricții folosind vectorul gradient. Aceste metode sunt cunoscute sub numele de metode de tip gradient. Dacă derivatele sunt continue și pot fi evaluate analitic, metodele de tip gradient sunt mai eficiente decât metodele de căutare de ordinul zero, cum ar fi metoda simplex, care folosesc numai evaluări ale funcției obiectiv. Metodele de tip gradient sunt recomandate pentru funcții cu derivate ușor de calculat analitic. În acest capitol sunt prezentate: metoda celei mai rapide coborri (“steepest descendent” cunoscută și sub numele de metoda gradientului) și metoda gradientilor conjugați.

7.1 Formularea problemei

Fie $f : \mathbb{R}^n \rightarrow \mathbb{R}$ o funcție reală de n variabile, cu derivatele de ordinul unu și doi continue. Se cere să se determine minimumul funcției f , respectiv punctul $\mathbf{x}^* \in \mathbb{R}^n$ astfel încât $f(\mathbf{x}^*) \leq f(\mathbf{x})$, pentru orice $\mathbf{x} \in \mathbb{R}^n$.

Dacă \mathbf{x}^* este un minim local al funcției f , atunci derivatele lui f față de variabilele independente, evaluate în \mathbf{x}^* sunt nule. În consecință, **condiția necesară** pentru ca \mathbf{x}^* să fie minim este ca vectorul gradient $\nabla f(\mathbf{x}^*) = \left[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right]^T$ să fie nul în \mathbf{x}^* .

Pentru o clasă largă de funcții **condiția suficientă** ca \mathbf{x}^* să fie minim este ca matricea Hessian a derivatelor de ordin doi $\nabla^2 f = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right] \neq 0$ să fie pozitiv definită în punctul \mathbf{x}^* (condiție îndeplinită cnd funcția f este convexă).

Exercițiul 7.1: Se dau funcțiile: $f_1(x, y) = x^2 + y^2$, $f_2(x, y) = 2x^2 + 4y^2$, $f_3(x, y) = x^2 - y^2$, $f_4(x, y) = x^2 - y$, $f_4 = |x| + |y|$, $f_5 = \frac{(x-1)^2}{4} + \frac{(y-2)^2}{16}$, $f_6(x, y) = x^4 + y^4$. Care dintre problemele de minimizare corespunzătoare acestor funcții sunt corect formulate?

7.2 Metoda celei mai rapide coborri (metoda gradientului)

7.2.1 Principiul metodei

Această metodă constă în generarea unui șir de direcții care corespund celei mai mari rate de schimbare a funcției f (direcției gradientului) și determinarea punctului de minim al funcției pe aceste direcții prin minimizări unidimensionale. Atunci cnd ea există, limita șirului de minime astfel obținute este minimul funcției f .

Să considerăm o funcție reală de două variabile $f(x, y)$. Dezvoltarea ei în serie Taylor în jurul punctului (x_0, y_0) este:

$$\begin{aligned} f(x, y) &= f(x_0, y_0) + \frac{\partial f}{\partial x}(x_0, y_0) \cdot (x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0) \cdot (y - y_0) + \\ &+ \frac{1}{2} \left[\frac{\partial^2 f}{\partial x^2}(x_0, y_0) \cdot (x - x_0)^2 + \frac{\partial^2 f}{\partial x \partial y}(x_0, y_0) \cdot (x - x_0)(y - y_0) + \right. \\ &+ \left. \frac{\partial^2 f}{\partial x \partial y}(x_0, y_0) \cdot (y - y_0)(x - x_0) + \frac{\partial^2 f}{\partial y^2}(x_0, y_0) \cdot (y - y_0)^2 \right] + \dots \quad (7.1) \end{aligned}$$

• Aproximarea pătratică (de ordinul doi) a funcției f

Dacă notăm:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{x}_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \\ f_0 &= f(x_0), \\ \mathbf{g}_0 &= \mathbf{g}(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial f}{\partial x}(\mathbf{x}_0) \\ \frac{\partial f}{\partial y}(\mathbf{x}_0) \end{bmatrix}, \\ \mathbf{H}_0 &= \mathbf{H}(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2}(\mathbf{x}_0) & \frac{\partial^2 f}{\partial x \partial y}(\mathbf{x}_0) \\ \frac{\partial^2 f}{\partial y \partial x}(\mathbf{x}_0) & \frac{\partial^2 f}{\partial y^2}(\mathbf{x}_0) \end{bmatrix}, \end{aligned}$$

unde \mathbf{g}_0 și \mathbf{H}_0 sunt gradientul, respectiv Hessianul funcției f , evaluate în punctul \mathbf{x}_0 , atunci, neglijnd termenii de ordin superior, relația (7.1) devine:

$$f(\mathbf{x}) \approx f_2(\mathbf{x}) = f_0 + (\mathbf{x} - \mathbf{x}_0)^T \mathbf{g}_0 + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}_0 (\mathbf{x} - \mathbf{x}_0). \quad (7.2)$$

Observație: Funcția f_2 dată de (7.2) reprezintă aproximarea pătratică (de ordin doi) a funcției f în vecinătatea punctului \mathbf{x}_0 . Ea are aceeași semnificație și pentru cazul în care domeniul de definiție al funcției este n -dimensional, \mathbf{g}_0 fiind gradientul funcției inițiale f evaluat în \mathbf{x}_0 , \mathbf{H}_0 fiind matricea Hessian evaluată în \mathbf{x}_0 .

Folosind aproximarea (7.2), gradientul funcției f are expresia:

$$\nabla f(\mathbf{x}) \approx \nabla f_2(\mathbf{x}) = \mathbf{g}_0 + \mathbf{H}_0(\mathbf{x} - \mathbf{x}_0). \quad (7.3)$$

Exercițiul 7.2: Verificați relația (7.3) pentru o funcție de două variabile. Indicație: folosind relația (7.1) calculați $\frac{\partial f}{\partial x}(x, y)$ și $\frac{\partial f}{\partial y}(x, y)$.

Dacă notăm $\mathbf{x} = \mathbf{x}_0 + \Delta\mathbf{x}$, aproximarea pătratică f_2 dată de relația (7.2) se scrie:

$$f_2(\mathbf{x}_0 + \Delta\mathbf{x}) = f_0 + \Delta\mathbf{x}^T \mathbf{g}_0 + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H}_0 \Delta\mathbf{x}, \quad (7.4)$$

iar relația (7.3) se transformă în:

$$\nabla f_2(\mathbf{x}_0 + \Delta\mathbf{x}) = \mathbf{g}_0 + \mathbf{H}_0 \Delta\mathbf{x}. \quad (7.5)$$

Dacă funcția f este minimă atunci gradientul ei este nul. Utiliznd în locul funcției f aproximarea sa pătratică f_2 , minimul se obține cu aproximație, atunci cnd

$$\nabla f_2 = 0 \iff \mathbf{H}_0 \Delta\mathbf{x} + \mathbf{g}_0 = \mathbf{0}. \quad (7.6)$$

Deci, minimul aproximării pătratice f_2 a funcției f este $\mathbf{x}_0 + \Delta\mathbf{x}$, unde $\Delta\mathbf{x} = -\mathbf{H}_0^{-1} \mathbf{g}_0$ este soluția sistemului liniar $\mathbf{H}_0 \Delta\mathbf{x} = -\mathbf{g}_0$.

Metoda iterativă bazată pe șirul:

$$\mathbf{x}_{k+1} - \mathbf{x}_k = -\mathbf{H}_k^{-1} \mathbf{g}_k, \quad (7.7)$$

este cunoscută sub numele de metoda Newton. În relația (7.7) \mathbf{x}_k reprezintă soluția la pasul k , $\mathbf{H}_k = \mathbf{H}(\mathbf{x}_k)$ reprezintă matricea Hessian evaluată în \mathbf{x}_k , iar $\mathbf{g}_k = \mathbf{g}(\mathbf{x}_k)$ este gradientul funcției f , evaluat în \mathbf{x}_k . Metoda este dezavantajoasă deoarece presupune pe de o parte calculul matricei Hessian (deci derivate de ordinul doi) la fiecare iterație iar pe de altă parte rezolvarea unui sistem liniar de n ecuații cu n necunoscute.

• Aproximarea liniară (de ordinul unu) a funcției f

Dacă neglijăm și termenii de ordinul doi în relația (7.2), rezultă:

$$f(\mathbf{x}) \approx f_1(\mathbf{x}) = f_0 + (\mathbf{x} - \mathbf{x}_0)^T \mathbf{g}_0 \quad (7.8)$$

sau

$$f_1(\mathbf{x}_0 + \Delta\mathbf{x}) = f_0 + \Delta\mathbf{x}^T \mathbf{g}_0, \quad (7.9)$$

astfel, variația funcției f între \mathbf{x} și \mathbf{x}_0 se poate aproxima cu:

$$f(\mathbf{x}) - f(\mathbf{x}_0) \approx f_1(\mathbf{x}) - f(\mathbf{x}_0) = \Delta \mathbf{x}^T \mathbf{g}_0. \quad (7.10)$$

În consecință, într-o aproximație de ordinul unu, o deplasare $\Delta \mathbf{x}$ a punctului curent conduce la o variație a funcției obiectiv:

$$\Delta f_1 = \Delta \mathbf{x}^T \mathbf{g}_0 = |\Delta \mathbf{x}| |\mathbf{g}_0| \cos \theta, \quad (7.11)$$

unde θ este unghiul dintre vectorii $\Delta \mathbf{x}$ și \mathbf{g}_0 .

Pentru ca metoda de determinare a minimului să fie ct mai rapid convergentă, se dorește o scădere ct mai mare a funcției obiectiv care se obține (la valori constante ale modulelor $|\mathbf{g}_0|$ și $|\Delta \mathbf{x}|$) pentru $\cos \theta = -1 \iff \theta = \pi$, deci, cnd vectorul $\Delta \mathbf{x}$ are direcția vectorului gradient, și sens opus acestuia:

$$\Delta \mathbf{x} = -\alpha \mathbf{g} = \alpha \mathbf{v}, \quad (7.12)$$

cu α un scalar pozitiv. Vectorul $\mathbf{v} = -\mathbf{g}$ reprezintă direcția de căutare, sensul său fiind opus, în orice punct \mathbf{x} , sensului gradientului $\mathbf{g}(\mathbf{x})$ în acel punct.

Metoda gradientului este o metodă iterativă, bazată pe relația (7.12). Pornind de la o inițializare \mathbf{x}_0 , metoda construiește un șir de soluții \mathbf{x}_k :

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k = \mathbf{x}_k + \alpha_k \mathbf{v}_k, \quad (7.13)$$

unde

$$\mathbf{v}_k = -\mathbf{g}_k = -\mathbf{g}(\mathbf{x}_k). \quad (7.14)$$

Valoarea optimă a coeficientului α_k se alege astfel înct ea să asigure minimizarea funcției de variabilă α : $t(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{v}_k)$ pe direcția $\mathbf{v}_k = -\mathbf{g}_k$. Această valoare optimă se determină prin căutare liniară (minimizare cu un singur parametru) pe direcția de căutare \mathbf{v}_k .

Observații:

1) Dacă minimizarea liniară este exactă, atunci direcțiile consecutive din metoda gradientului sunt perpendiculare, deoarece minimizarea liniară conduce la un punct în care derivata funcției după direcția după care a avut loc minimizarea este zero. În consecință gradientul în punctul nou obținut este perpendicular pe direcția după care a avut loc minizarea, iar minimizarea următoare are loc tocmai după direcția acestui gradient. Putem scrie deci că

$$\mathbf{g}_{k+1}^T \mathbf{v}_k = \mathbf{v}_k^T \mathbf{g}_{k+1} = 0. \quad (7.15)$$

2) Căutarea liniară care conduce la valoarea optimă α_k a parametrului α s-ar putea elimina dacă se cunoaște Hessianul H și se presupune funcția f aproximată prin aproximarea ei de ordinul doi f_2 . Prin egalarea cu zero a derivatei în raport cu α a funcției $t_2(\alpha) = f_2(\mathbf{x}_k + \alpha \mathbf{v}_k)$, cu $f_2(\mathbf{x})$ dată de relația (7.4) și $\mathbf{v}_k = -\mathbf{g}_k$, se obține:

$$\alpha'_k = -\frac{\mathbf{v}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k} = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k}. \quad (7.16)$$

Exercițiul 7.3: Demonstrați relația (7.16).

Figura 7.1 prezintă variația cu α a funcției f și a aproximărilor sale pe direcția \mathbf{v}_k . Se constată că folosind aproximarea de ordinul doi, valoarea α'_k obținută cu relația (7.16) nu este cea optimă, α_k , valoare care se poate obține doar prin căutare liniară.

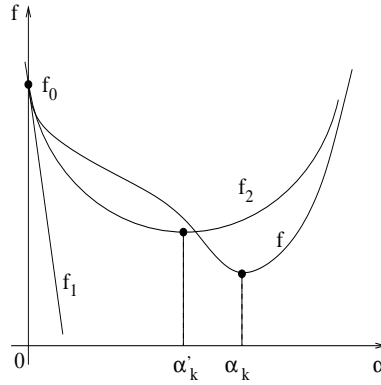


Figura 7.1: Funcția f și aproximările sale pe direcția \mathbf{v}_k .

3) Condiția de oprire poate fi implementată în mai multe feluri. Algoritmul se poate termina atunci cnd: valoarea funcției nu s-a schimbat prea mult la ultimele două iterații, poziția punctului curent nu s-a modificat semnificativ, sau gradientul funcției s-a apropiat suficient de mult de zero:

- **condiția de oprire pentru valoarea funcției de minimizat:**

$$\frac{|f_{k+1} - f_k|}{1/2(|f_{k+1}| + |f_k| + \text{EPS})} < \text{ftol}. \quad (7.17)$$

În relația (7.17), parametrul EPS reprezintă un estimator al erorii relative de rotunjire a mașinii (zeroul mașinii), introdus în formulă pentru a evita împărțirea la zero.

- **condiția de oprire pentru valoarea pasului:**

$$\frac{|\mathbf{x}_{k+1} - \mathbf{x}_k|}{|\mathbf{x}_1 - \mathbf{x}_0|} < \text{xtol}. \quad (7.18)$$

- condiția de oprire pentru gradient:

$$|\mathbf{g}_{k+1}| < \text{gtol}. \quad (7.19)$$

7.2.2 Algoritmul metodei

Considerentele teoretice din paragraful anterior conduc la următorul algoritm general pentru metoda gradientului:

1. Alege \mathbf{x}_0
 $k = 0$
 Calculează $\mathbf{g}_0 = \nabla f(x_0)$
 2. **repetă**
 - 2.1. $\mathbf{v}_k = -\mathbf{g}_k$
 - 2.2. Minimizează $t(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{v}_k)$ în raport cu $\alpha \Rightarrow \alpha_k$
 - 2.3. $\mathbf{p}_k = \alpha_k \mathbf{v}_k$
 - 2.4. $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$
 - 2.5. Calculează $\mathbf{g}_{k+1} = \nabla f(x_{k+1})$
 - 2.6 $k = k + 1$
- până când** este îndeplinită condiția de oprire

În implementarea acestui algoritm, pentru a evita irosirea memoriei, variabilele nu vor fi indexate cu contorul de iterații k , $k + 1$.

Exercițiul 7.4: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere corespunzătoare acestei teme.

b) Descrieți conținutul acestor fișiere.

c) Executați programul **main_gradienti.sci**. Programul minimizează o funcție prin metoda gradientului și afișează soluția, valoarea funcției în soluție, numărul de iterații și timpul de calcul. Programul reprezintă grafic evoluția soluției, unind prin linii punctele succesive. Care este expresia funcției minimizată de program? Cum este implementată condiția de oprire?

7.2.3 Efort de calcul

Chiar și pentru o funcție pătratică¹ această metodă este inefficientă din punct de vedere al efortului de calcul, fiind necesar un număr de pași mult mai mare decât numărul de variabile ale funcției. În figura 7.2 sunt desenate curbele de nivel (izovalori sau echivalori) ale unei funcții de două variabile ($n = 2$) și este reprezentată traiectoria iterațiilor în procesul de determinare al minimumului.

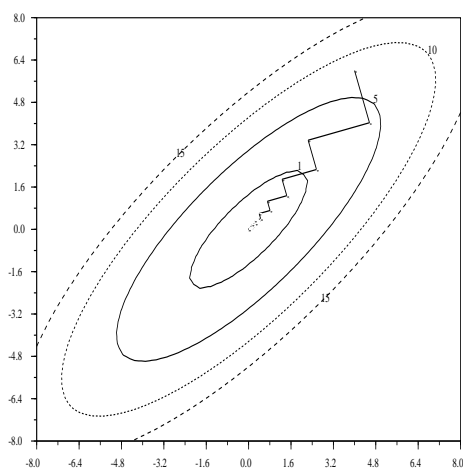


Figura 7.2: Evoluția soluției în metoda gradientului.

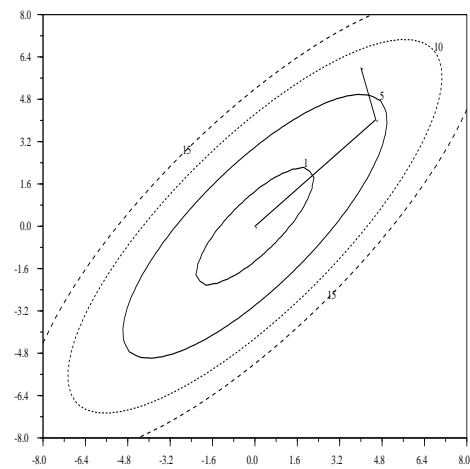


Figura 7.3: Evoluția soluției în metoda gradientilor conjugați.

Exercițiul 7.5: Fie funcția pătratică

$$f(x, y) = \frac{(x \cos \theta - y \sin \theta)^2}{A^2} + \frac{(x \sin \theta + y \cos \theta)^2}{B^2}. \quad (7.20)$$

- Ce reprezintă graficul funcției? Considerați cazurile $A = B$ și $A \neq B$.
- Ce reprezintă izovalorile funcției? Considerați cazurile $A = B$ și $A \neq B$.
- Cum afectează graficul și izovalorile funcției modificarea parametrilor A și B ?
- Cum afectează graficul și izovalorile funcției modificarea parametrului θ ?

¹O funcție pătratică în n dimensiuni are forma $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c$, unde A este o matrice pătrată de dimensiune n , \mathbf{b} este un vector coloană de dimensiune n iar c este o constantă reală. Termenul $\frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x}$ generează produse de gradul doi de tipul $\frac{1}{2}(a_{ij} + a_{ji})x_i x_j$, iar termenul $\mathbf{b}^T \mathbf{x}$ generează termeni de gradul unu $b_k x_k$. Fără a micșora generalitatea se poate presupune că matricea A este simetrică.

e) Calculați gradientul și matricea Hessian pentru funcția f . Identificați în program funcțiile care calculează gradientul și Hessianul acestei funcții.

f) Pentru $A = 1$ și $B = 3$ notați numărul de iterații efectuate și timpul de calcul pentru următoarele valori ale parametrului θ : $0, \pi/6, \pi/4, \pi/3, \pi/2, 2\pi/3, 3\pi/4, 5\pi/6, \pi$. Comentați rezultatele.

g) Rulați programul pentru $A = B = 1$. Observați câte iterații sunt necesare și explicați rezultatul. Ce influență are parametrul θ asupra procesului de minimizare?

h) Cum se face minimizarea liniară?

i) Studiați influența următorilor parametri asupra efortului de calcul, în cazul $A = 1, B = 3, \theta = \pi/3$:

- Toleranța impusă funcției (în relația (7.17)). Alegeți $\text{ftol} = 10^{-15}, 10^{-10}, 10^{-6}, 10^{-3}, 10^{-1}$;
- Estimatorul EPS al erorii relative (în relația (7.17)). Alegeți: $\text{EPS} = \text{zeroul mașinii}, 10^{-10}, 10^{-5}, 10^{-2}, 10^{-1}, 0$;
- Inițializarea \mathbf{x}_0 . Alegeți $\mathbf{x}_0 = (2, 2), (2, -2), (-2, 2), (-2, -2), (1, 1), (1, -1), (-1, 1), (-1, -1)$.

Exercițiul 7.6: După cum ați observat, calculul minimului după o direcție se face exact în cazul exercițiului 7.5. Programul **main_grad.sci** minimizează de asemenea funcția de la exercițiul 7.5 folosind pentru minimizarea liniară metoda secțiunii de aur prezentată la tema 2.

a) Comparați parametrii funcției ce implementează metoda secțiunii de aur cu cei ai aceleași funcții prezentate la tema 2. Comentați diferențele.

b) Pentru $\theta = \pi/3, A = 1, B = 3$, studiați influența parametrilor metodei secțiunii de aur asupra efortului de calcul al întregului proces:

- Limitele domeniului inițial în metoda secțiunii de aur (la și lb). Pentru $\text{eps_aur} = 10^{-17}$ fixat, alegeți următoarele perechi (la, lb): $(-1, 1), (-1000, 1000)$;
- Eroarea impusă în metoda secțiunii de aur. Pentru $(\text{la}, \text{lb}) = (-1000, 1000)$, alegeți $\text{eps_aur} = \text{zeroul mașinii}, 10^{-17}, 10^{-6}, 10^{-3}, 10^{-1}, 1.6$;

c) De ce direcțiile succesive de căutare nu sunt întotdeauna perpendiculare?

Exercițiul 7.7: a) Modificați programul **main_grad.sci** astfel încât să se minimizeze funcția lui Rosenbrock ("banană"). Funcția este descrisă la tema 5.

b) Studiați influența parametrilor metodei gradientului și metodei secțiunii de aur în cazul acestei funcții. Comparați rezultatele cu cele obținute în cazul aceluiași studiu făcut pentru funcția (7.20).

7.3 Metoda gradientilor conjugați

7.3.1 Principiul metodei

Metoda gradientilor conjugați este tot o metodă iterativă de căutare unidimensională bazată pe relația (7.13). La această metodă însă, direcția de căutare depinde att de direcția gradientului, ct și de direcția de căutare anterioară:

$$\mathbf{v}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{v}_k, \quad k > 0, \quad (7.21)$$

(comparați cu (7.14)), iar

$$\mathbf{v}_0 = -\mathbf{g}_0.$$

Exercițiul 7.8: Prin ce diferă primul pas al metodei gradientilor conjugați de primul pas al metodei gradientului?

Metoda se numește astfel deoarece direcțiile \mathbf{v}_k sunt alese astfel încât să fie **conjugate** (mai exact **H-conjugate** sau **H-ortogonale**):

$$\mathbf{v}_{k+1}^T \mathbf{H}_k \mathbf{v}_j = 0, \quad (\forall) j \leq k. \quad (7.22)$$

Condiția (7.22) scrisă pentru cazul particular $j = k$ permite determinarea coeficientului β_k :

$$\begin{aligned} \mathbf{v}_{k+1}^T \mathbf{H}_k \mathbf{v}_k = 0 & \iff (-\mathbf{g}_{k+1}^T + \beta_k \mathbf{v}_k^T) \mathbf{H}_k \mathbf{v}_k = 0 \implies \\ \beta_k &= \frac{\mathbf{g}_{k+1}^T \mathbf{H}_k \mathbf{v}_k}{\mathbf{v}_k^T \mathbf{H}_k \mathbf{v}_k}, \end{aligned} \quad (7.23)$$

relație care ar necesita cunoașterea Hessianului în \mathbf{x}_k .

Folosind aproximația de ordinul doi pentru funcția f și gradientul acesteia, relația (7.3) scrisă în $\mathbf{x} = \mathbf{x}_{k+1}$ și $\mathbf{x}_0 = \mathbf{x}_k$ devine:

$$\mathbf{g}(\mathbf{x}_{k+1}) - \mathbf{g}(\mathbf{x}_k) = \mathbf{g}_{k+1} - \mathbf{g}_k \approx \mathbf{H}_k \Delta \mathbf{x}_k, \quad (7.24)$$

și cu folosirea relației iterative:

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \Delta \mathbf{x}_k = \alpha_k \mathbf{v}_k, \quad (7.25)$$

coeficientul β_k poate fi determinat astfel:

$$\begin{aligned}\beta_k &= \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{v}_k^T(\mathbf{g}_{k+1} - \mathbf{g}_k)} \\ &= \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{(-\mathbf{g}_k + \beta_{k-1} \mathbf{v}_{k-1})^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}.\end{aligned}$$

Exercițiul 7.9: În ce caz relația (7.24) este exactă?

Deoarece, după cum se poate arăta, șirul \mathbf{x}_k este astfel construit încât gradientul în \mathbf{x}_k și direcția \mathbf{v}_k satisfac în plus relațiile de ortogonalitate:

$$\mathbf{g}_{k+1}^T \mathbf{g}_j = 0, \quad \mathbf{v}_{k+1}^T \mathbf{g}_j = 0, \quad j \leq k, \quad (7.26)$$

coeficientul β_k devine:

$$\beta_k = \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k} = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}. \quad (7.27)$$

Oricare din cele două formule din relația (7.27) se pot folosi. Fiecare din ele este cunoscută sub un nume celebru, și anume:

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \quad - \text{ formula Fletcher-Reeves} \quad (7.28)$$

$$\beta_k = \frac{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \quad - \text{ formula Polak-Ribiere.} \quad (7.29)$$

Observații:

1) Deși relațiile (7.28) și (7.29) sunt echivalente în aritmetica exactă deoarece $\mathbf{g}_{k+1}^T \mathbf{g}_k = 0$, totuși, în rezolvarea cu ajutorul calculatorului, datorită erorilor de calcul este posibil ca direcțiile \mathbf{g}_k să nu fie ortogonale, de aceea formula Polak-Ribiere (7.29) este de preferat în implementarea numerică, ea compensnd eventualele mici “defecte de ortogonalitate”.

2) Această metodă se încadrează în clasa, mai largă, a metodelor de căutare în care direcția la pasul curent depinde att de direcția gradientului în punctul curent, ct și de **toate** direcțiile de căutare anterioare:

$$\mathbf{v}_{k+1} = -\mathbf{g}_{k+1} + \sum_{j=0}^k C_{j,k+1} \mathbf{v}_j, \quad k = 0, \dots, n-1, \quad (7.30)$$

(comparați cu (7.21)).

Condițiile de ortogonalitate (7.26) fac ca, în relația (7.30), singurul coeficient nenul $C_{j,k+1}$ să fie acela pentru $j = k$, $C_{k,k+1} = \beta_k$.

Exercițiul 7.10: Determinați expresia coeficientului $C_{j,k+1}$ din (7.30). Verificați că, dacă sunt îndeplinite condițiile de ortogonalitate (7.26), singurul coeficient nenul este $C_{k,k+1}$.

7.3.2 Algoritmul metodei

Algoritmul iterativ al metodei gradientilor conjugați în varianta Polak-Ribiere este prezentat în următorul pseudocod, în care s-au păstrat indicii $(0, k, k + 1)$ pentru facilitarea înțelegerii:

1. Alege \mathbf{x}_0
 $k = 0$
 Calculează \mathbf{g}_0
 $\beta = 0$
2. **repetă**
 - 2.1. $\mathbf{v}_k = -\mathbf{g}_k + \beta \mathbf{v}_{k-1}$
 - 2.2. Minimizează $t(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{v}_k)$ în raport cu $\alpha \Rightarrow \alpha_k$
 - 2.3. $\mathbf{p}_k = \alpha_k \mathbf{v}_k$
 - 2.4. $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$
 - 2.5. Calculează \mathbf{g}_{k+1}
 - 2.6. $\beta = (\mathbf{g}_{k+1} - \mathbf{g}_k)^T * \mathbf{g}_{k+1} / (\mathbf{g}_k^T * \mathbf{g}_k)$;
 - 2.7. $k = k + 1$

până când este îndeplinită condiția de oprire

Exercițiul 7.11: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere corespunzătoare metodei gradientilor conjugați.

b) Descrieți conținutul acestor fișiere. Ce diferențe de implementare există între programul **main_gradienti_cj.sci** și programul **main_gradienti.sci** corepunzător metodei gradientului?

c) Executați programul **main_gradienti.sci**, care minimizează funcția dată de relația (7.20) prin metoda gradientilor conjugați și afișează soluția, valoarea funcției în soluție, numărul de iterații și timpul de calcul. De asemenea, programul reprezintă grafic evoluția soluției, unind prin linii punctele succesive. Cum se realizează minimizarea liniară?

7.3.3 Efort de calcul

Folosind metoda gradientilor conjugați, pentru o funcție pătratică de n variabile se demonstrează că minimul se obține după n pași (într-o aritmetică exactă). De aceea se spune că metoda gradientilor conjugați este o metodă semiiterativă. Figura 7.3 prezintă evoluția soluției în metoda gradientilor conjugați pentru aceeași funcție și aceeași inițializare ca pentru metoda gradientului reprezentată în figura 7.2.

Metoda gradientilor conjugați este în general mai rapid convergentă decât metoda gradientului pentru funcții multidimensionale mai complexe (funcții neparabolice).

Exercițiul 7.12: a) Modificați programul `main_gradienti_cj.sci` pentru a minimiza funcția definită la exercițiul 5, cu $A = 1$, $B = 3$, pentru diverse valori ale parametru-ului θ : $0, \pi/6, \pi/3, \pi/2, 2\pi/3, 3\pi/4, 5\pi/6, \pi$. Notați, pentru fiecare valoare a unghiului, numărul de iterații efectuate și timpul de calcul. Comentați rezultatul.

b) Studiați influența parametrilor `ftol` și `EPS` precum și influența punctului inițial, așa cum ați făcut la metoda gradientului, la exercițiul 7.5, punctul i.

Exercițiul 7.13: Programul `main_grad_cj.sci` minimizează de asemenea funcția (7.20) dar folosește pentru minimizarea liniară metoda secțiunii de aur. Studiați influența parametrilor metodei secțiunii de aur asupra efortului de calcul al întregului proces așa cum ați făcut la exercițiul 7.6, punctul b.

Exercițiul 7.14: Modificați programul `main_grad_cj.sci` astfel încât să se minimizeze funcția lui Rosenbrock prin metoda gradientilor conjugați. Studiați influența parametrilor metodei gradientilor conjugați și metodei secțiunii de aur în cazul acestei funcții. Comparați rezultatele cu cele obținute în cazul aceleiași studiu făcut pentru funcția (7.20). Comparați rezultatele cu cele obținute la metoda gradientului.

Exercițiul 7.15: Scrieți un program care să minimizeze funcția de n variabile

$$f(\mathbf{x}) = \sum_{i=1}^n (x_i - i)^2 \quad (7.31)$$

prin metoda gradientului și metoda gradientilor conjugați. Programul va permite introducerea de la tastatură a lui n și alegerea metodei de calcul. Studiați experimental cum depinde efortul de calcul de dimensiunea problemei

7.4 Aspecte legate de convergență

Pentru o funcție cu derivate continue, metoda gradientului converge către un punct staționar, strategia fiind bazată pe aproximarea în serie Taylor de ordinul unu a funcției de minimizat. Metoda gradientului este deci foarte înceată în apropierea minimului deoarece direcțiile succesive de căutare sunt ortogonale (vezi figura 7.2).

O metodă care converge ca metoda gradientului se spune că are o **convergență liniară**. Aceasta înseamnă că $\|\mathbf{x}_{k+1} - \mathbf{x}_{\min}\|/\|\mathbf{x}_k - \mathbf{x}_{\min}\|$ converge către o constantă pozitivă β mai mică decât 1, numită *rată de convergență*. În cazul metodei gradientului, rata de convergență depinde de matricea Hessian \mathbf{H}_{\min} în punctul de minim. Pentru o funcție cu derivate parțiale de ordinul doi continue, \mathbf{H}_{\min} trebuie să fie pozitiv definită, adică valorile sale proprii trebuie să fie numere reale pozitive. Dacă M și m sunt cea mai

mare și, respectiv, cea mai mică valoare proprie, se poate arăta că rata de convergență este dată de formula

$$\beta = \left(\frac{M - m}{M + m} \right)^2 = \left(\frac{r - 1}{r + 1} \right)^2, \quad (7.32)$$

unde $r = M/m$ este raportul dintre cea mai mare și cea mai mică valoare proprie a matricei Hessian în punctul de minim, număr care se numește și număr de condiționare al matricei.

Convergența liniară este numită uneori și **convergență geometrică** deoarece se poate arăta că pentru k' suficient de mare este adevărată relația

$$\|\mathbf{x}_k - \mathbf{x}_{\min}\| \approx \beta^{k-k'} \|\mathbf{x}_{k'} - \mathbf{x}_{\min}\|. \quad (7.33)$$

Observații:

1. Dacă r tinde către 1, atunci β tinde către 0. Un algoritm care are proprietatea că $\|\mathbf{x}_{k+1} - \mathbf{x}_{\min}\| / \|\mathbf{x}_k - \mathbf{x}_{\min}\|$ tinde către 0 se spune că are **convergența superliniară**.
2. Cu cât matricea \mathbf{H}_{\min} este mai prost condiționată, adică cu cât r este mai mare decât 1, cu atât metoda gradientului converge mai greu. Aceasta este fundamentarea teoretică a faptului că se practică scalarea variabilelor astfel încât derivatele funcției f să aibă aceleași ordine de mărime.
3. Dacă $\|\mathbf{x}_{k+1} - \mathbf{x}_{\min}\| / \|\mathbf{x}_k - \mathbf{x}_{\min}\|^2$ converge către o constantă se spune că algoritmul are o convergență pătratică (sau de ordinul doi). În general se spune că un algoritm are **ordinul de convergență** p dacă $\|\mathbf{x}_{k+1} - \mathbf{x}_{\min}\| / \|\mathbf{x}_k - \mathbf{x}_{\min}\|^p$ converge către o constantă. Cazul $p > 2$ este rareori întâlnit.

Pe scurt:

- Metoda gradientului are convergență liniară;
- Metoda gradientilor conjugați are convergență superliniară după n pași;
- Metoda Newton are convergență pătratică.

Exercițiul 7.16: *Comparați experimental cele două metode studiate în această lucrare (metoda gradientului și metoda gradientilor conjugați) din punctul de vedere al vitezei de convergență. Indicații: notați valorile $\|\mathbf{x}_{k+1} - \mathbf{x}_{\min}\| / \|\mathbf{x}_k - \mathbf{x}_{\min}\|$, calculați matricea Hessian în punctul de minim, valorile ei proprii (folosiți comanda Scilab `spec`) și rata de convergență, etc.*

Capitolul 8

Minimizări multidimensionale - metode deterministe de ordinul unu. Metode quasi-Newton.

Din clasa metodelor deterministe de ordinul unu fac parte și metodele numite *quasi-Newton*. Ele sunt numite astfel deoarece încearcă să simuleze iterații de tip Newton-Raphson, plasându-se cumva între metoda gradientului și metoda Newton. Metoda Newton necesită evaluarea inversei matricei Hessian, lucru care este foarte costisitor din punct de vedere numeric. Ca urmare, a apărut ideea de a lucra cu o aproximare a inversei matricei Hessian calculată cu ajutorul vectorului gradient evaluat în iterațiile precedente, idee care stă la baza metodelor quasi-Newton. Variantele metodelor de tip quasi-Newton diferă prin felul în care se face această aproximare. Aproximările pot fi din cele mai simple, în care matricea aproximativă rămâne constantă pe parcursul iterațiilor, până la cele mai avansate, în care se construiesc aproximații din ce în ce mai bune ale inversei matricei Hessian, pe baza informațiilor adunate în timpul procesului de coborâre. Această din urmă abordare corespunde metodelor din clasa algoritmilor de *metrică variabilă*.

Prima și una din cele mai importante scheme de construcție a inversei matricei Hessian a fost propusă de Davidon (1959). Metoda a fost mai târziu modificată și îmbunătățită de Fletcher și Powell (1964), algoritmul propus de ei fiind cunoscut sub numele de algoritmul **Davidon-Fletcher-Powell** (DFP). O altă variantă este cunoscută sub numele **Broyden-Fletcher-Goldfarb-Shanno** (BFGS). Algoritmii DFP și BFGS diferă numai în detalii legate de eroarea de rotunjire, toleranțele de convergență și alte aspecte de acest tip. Totuși, a devenit în general recunoscut că, empiric, schema BFGS este superioară din punctul de vedere al acestor detalii. Acest capitol prezintă acești doi algoritmi. Toate notațiile care intervin precum și formularea problemei sunt aceleași ca cele din capitolul 7.

8.1 Metoda Newton modificată

Conform relației (7.7) metoda Newton se bazează pe șirul de iterații $\mathbf{x}_{k+1} - \mathbf{x}_k = -\mathbf{H}_k^{-1} \mathbf{g}_k$. În continuare vom studia un proces iterativ care urmărește minimizarea funcției $f(\mathbf{x})$ și este generat de relația

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{S}_k \mathbf{g}_k, \quad (8.1)$$

unde \mathbf{x}_k este un vector coloană de dimensiune n care reprezintă soluția numerică la iterația k , \mathbf{g}_k este un vector coloană de dimensiune n care reprezintă gradientul funcției f evaluat în punctul \mathbf{x}_k , \mathbf{S}_k este o matrice simetrică de dimensiune $n \times n$ și α_k este un număr real pozitiv ales astfel încât $t(\alpha) = f(\mathbf{x}_{k+1})$ să fie minimă¹. Dacă $\alpha_k \mathbf{S}_k$ este inversa matricei Hessian a lui f atunci relația (8.1) este, conform relației (7.7), cea corespunzătoare metodei Newton. Dacă $\mathbf{S}_k = \mathbf{I}$ atunci relația (8.1) este cea corespunzătoare metodei celei mai rapide coborâri (vezi și relațiile (7.13) și (7.14)).

Strategia constă în a alege pentru \mathbf{S}_k o aproximație a inversei matricei Hessian. Această strategie ar putea fi mai bună decât cea din metoda Newton în care se folosește inversa exactă a matricei Hessian. Pentru a putea înțelege această afirmație, să considerăm direcțiile \mathbf{v} care trec prin \mathbf{x}_k . Pentru ca \mathbf{v} să fie o direcție de-a lungul căreia funcția f să descrească trebuie ca $\mathbf{v}^T \mathbf{g}_k < 0$. De aceea, pentru ca direcția dată de $\mathbf{x}_{k+1} - \mathbf{x}_k$ să fie o direcție de-a lungul căreia funcția f să descrească trebuie ca $(\mathbf{x}_{k+1} - \mathbf{x}_k)^T \mathbf{g}_k < 0$. În cazul metodei Newton, această condiție este echivalentă cu $-(\mathbf{x}_{k+1} - \mathbf{x}_k)^T \mathbf{H}_k (\mathbf{x}_{k+1} - \mathbf{x}_k) < 0$, adică matricea \mathbf{H}_k trebuie să fie pozitiv definită. Făcând pasul Newton, cu matricea Hessian exactă, nu există garanția că matricea Hessian este pozitiv definită, cum se întâmplă când suntem departe de minim. Am putea ajunge astfel într-un punct în care funcția crește ca valoare. Pe de altă parte, execuția unui pas Newton chiar cu o matrice pozitiv definită s-ar putea să nu conducă la o descreștere a funcției pentru că aproximația de ordinul doi s-ar putea să nu fie validă la depărtare de minim (vezi paragraful 7.2.1).

Exercițiul 8.1: Cum trebuie să fie matricea \mathbf{H}_k în cazul problemelor de maximizare pentru ca, după direcția $\mathbf{x}_{k+1} - \mathbf{x}_k$, funcția f să crească?

Făcând un raționament similar în cazul metodei Newton modificate, rezultă că matricea \mathbf{S}_k trebuie să fie pozitiv definită pentru a asigura ca procesul dat de relația (8.1) să fie întotdeauna unul coborâtor.

Exercițiul 8.2: Demonstrați această afirmație.

De aceea, în cele ce urmează vom impune întotdeauna ca \mathbf{S}_k să fie o matrice pozitiv definită. În acest fel se garantează că, chiar și departe de minim, ne mișcăm întotdeauna

¹ Algoritmul dat de relația (8.1) este cunoscut și sub numele de *metoda gradientilor deviați*, deoarece vectorul direcție se obține printr-o transformare liniară a gradientului (prin înmulțirea lui cu matricea \mathbf{S}_k).

într-o direcție în care funcția descrește. În apropierea minimului, formula aproximativă se apropie de formula exactă și vom obține convergența pătratică a metodei Newton.

Metodele quasi-Newton se deosebesc între ele prin modul de alegere a aproximării matricei \mathbf{S}_k . În **metoda Newton modificată clasică** matricea Hessian în punctul inițial \mathbf{x}_0 este folosită pe tot parcursul procesului iterativ. Succesul acestei metode depinde de cât de mult variază matricea Hessian, deci depinde de ordinul de mărime al derivatelor de ordinul trei ale funcției f .

Printre cele mai de succes metode quasi-Newton sunt cele care fac parte din clasa algoritmilor numiți de **metrică variabilă**. Trăsătura comună a acestora este aceea că estimarea inversei matricei Hessian este calculată în permanență folosind informațiile care se referă la modificările vectorului gradient, informații adunate în timpul procesului iterativ. Aceste metode vor fi descrise în cele ce urmează.

8.2 Construcția inversei matricei Hessian. Corecția de rangul unu.

În acest paragraf vom arăta cum se poate construi inversa matricei Hessian din informațiile legate de gradient în diferite puncte. Ideal este ca aproximarea \mathbf{S}_k să convergă către inversa matricei Hessian în punctul soluție și metoda să se comporte global ca metoda Newton.

Considerăm două puncte consecutive \mathbf{x}_k și \mathbf{x}_{k+1} și notăm cu \mathbf{g}_k și \mathbf{g}_{k+1} vectorii gradient în aceste puncte. Folosind aproximarea de ordinul doi a funcției f în vecinătatea punctului \mathbf{x}_k se obține, conform relației (7.3),

$$\mathbf{g}_{k+1} \approx \mathbf{g}_k + \mathbf{H}_k(\mathbf{x}_{k+1} - \mathbf{x}_k). \quad (8.2)$$

Pentru a simplifica scrierea relațiilor în justificările ce urmează, vom nota:

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \quad (8.3)$$

$$\mathbf{q}_k = \mathbf{g}_{k+1} - \mathbf{g}_k. \quad (8.4)$$

Dacă matricea Hessian este o matrice constantă $\mathbf{H}_k = \mathbf{H}$ (lucru adevărat în cazul în care funcția f este pătratică), atunci relația (8.2) devine

$$\mathbf{q}_k = \mathbf{H}\mathbf{p}_k, \quad (8.5)$$

relație care arată faptul că evaluarea gradientului în două puncte dă informații despre matricea Hessian \mathbf{H} . Dacă se consideră n direcții liniar independente $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}$ și dacă vectorii corespunzători \mathbf{q}_k sunt cunoscuți, atunci \mathbf{H} este determinată în mod unic.

Este natural să încercăm să construim aproximații succesive \mathbf{S}_k ale inversei matricei Hessian bazate pe datele obținute din primii k pași ai procesului de coborâre astfel încât, dacă \mathbf{H} ar fi constantă, aproximația să satisfacă relația (8.5), adică:

$$\mathbf{S}_{k+1}\mathbf{q}_i = \mathbf{p}_i, \quad 0 \leq i \leq k. \quad (8.6)$$

După n pași liniari independenți se obține $\mathbf{S}_n = \mathbf{H}^{-1}$.

Pentru orice $k < n$, problema construirii unei aproximări potrivite \mathbf{S}_k a inversei matricei Hessian admite o infinitate de soluții deoarece sunt mai multe grade de libertate decât restricții. De aceea, o metodă anume folosește și alte considerații.

Vom descrie acum una din cele mai simple scheme care a fost propusă, numită **corecția de rangul unu**.

Deoarece \mathbf{H} și \mathbf{H}^{-1} sunt simetrice, este natural să se construiască o aproximație \mathbf{S}_k a lui \mathbf{H}^{-1} care este de asemenea simetrică. Vom investiga posibilitatea definirii unei scheme recursive de forma:

$$\mathbf{S}_{k+1} = \mathbf{S}_k + \mathbf{z}_k \mathbf{z}_k^T, \quad (8.7)$$

care păstrează simetria. Vectorul coloană \mathbf{z}_k definește o matrice care are rangul cel mult unu și care corectează aproximarea inversei matricei Hessian. Le vom alege astfel încât relația (8.6) să fie satisfăcută. Luând i egal cu k în relația (8.6) și folosind relația (8.7) se obține:

$$\mathbf{p}_k = \mathbf{S}_{k+1}\mathbf{q}_k = \mathbf{S}_k\mathbf{q}_k + \mathbf{z}_k \mathbf{z}_k^T \mathbf{q}_k, \quad (8.8)$$

de unde rezultă că:

$$\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k = \mathbf{z}_k \mathbf{z}_k^T \mathbf{q}_k, \quad (8.9)$$

$$(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)^T = \mathbf{q}_k^T \mathbf{z}_k \mathbf{z}_k^T, \quad (8.10)$$

și înmulțind aceste două relații rezultă că:

$$\mathbf{z}_k \mathbf{z}_k^T = \frac{(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)^T}{(\mathbf{z}_k^T \mathbf{q}_k)^2}. \quad (8.11)$$

Pe de altă parte, înmulțind la stânga relația (8.9) cu \mathbf{q}_k^T (aceasta corespunde produsului scalar al celor doi vectori²), rezultă că:

$$\mathbf{q}_k^T (\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k) = (\mathbf{z}_k^T \mathbf{q}_k)^2. \quad (8.12)$$

Folosind relațiile (8.12) și (8.11), rezultă că relația (8.7) devine:

$$\mathbf{S}_{k+1} = \mathbf{S}_k + \frac{(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)^T}{\mathbf{q}_k^T (\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)}. \quad (8.13)$$

²Atenție: vectorii sunt considerați vectori coloană. Fie \mathbf{a} și \mathbf{b} doi vectori coloană de dimensiune n . Atunci $\mathbf{a}^T \mathbf{b}$ este un număr real iar $\mathbf{a} \mathbf{b}^T$ este o matrice pătrată de dimensiune $n \times n$. Uneori se mai folosește notația $\mathbf{a}^T \mathbf{b} = \mathbf{a} \cdot \mathbf{b}$ (produs scalar) și $\mathbf{a} \mathbf{b}^T = \mathbf{a} \otimes \mathbf{b}$ (produs diadic).

Am determinat deci corecția astfel încât relația (8.6) să fie satisfăcută pentru $i = k$. Se poate demonstra prin inducție matematică faptul că, în cazul în care matricea Hessian este constantă (deci pentru o funcție pătratică), relația (8.6) este satisfăcută de asemenea pentru $i < k$. Aceasta implică faptul că relația recursivă (8.13) face ca \mathbf{S}_k să convergă către \mathbf{H}^{-1} după cel mult n pași.

Pentru a incorpora aproximarea inversei matricei Hessian într-o procedură de coborâre care în același timp o îmbunătățește se procedează astfel: se calculează direcția $\mathbf{v}_k = -\mathbf{S}_k \mathbf{g}_k$, se minimizează apoi funcția $t(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{v}_k)$ în raport cu $\alpha \geq 0$. Aceasta va determina următorul punct $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{v}_k$. Se calculează apoi $\mathbf{p}_k = \alpha_k \mathbf{v}_k$, \mathbf{g}_{k+1} , \mathbf{q}_k . În cele din urmă \mathbf{S}_{k+1} se calculează cu formula (8.13). Algoritmul are următorul pseudocod, în care indexarea cu k și $k + 1$ a fost păstrată pentru a facilita înțelegerea.

1. Alege \mathbf{S}_0 simetrică și pozitiv definită

Alege \mathbf{x}_0

$k = 0$

Calculează $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$

2. **repetă**

2.1. $\mathbf{v}_k = -\mathbf{S}_k \mathbf{g}_k$

2.2. Minimizează $t(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{v}_k)$ în raport cu $\alpha \geq 0 \Rightarrow \alpha_k$

2.3. $\mathbf{p}_k = \alpha_k \mathbf{v}_k$

2.4. $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$

2.5. Calculează $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$

2.6. $\mathbf{q}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$

2.7. $\mathbf{S}_{k+1} = \mathbf{S}_k + (\mathbf{p}_k - \mathbf{S}_k \mathbf{q}_k)(\mathbf{p}_k - \mathbf{S}_k \mathbf{q}_k)^T / (\mathbf{q}_k^T (\mathbf{p}_k - \mathbf{S}_k \mathbf{q}_k))$

2.8 $k = k + 1$

până când este îndeplinită condiția de oprire

Există unele dificultăți legate de această procedură de rangul unu. În primul rând \mathbf{S}_{k+1} calculată cu formula (8.13) rămâne pozitiv definită numai dacă $\mathbf{q}_k^T (\mathbf{p}_k - \mathbf{S}_k \mathbf{q}_k) > 0$, condiție care nu poate fi garantată. De asemenea, chiar dacă această mărime este pozitivă, în cazul în care ea este mică apar dificultăți numerice. Astfel, deși metoda de rangul unu este un excelent exemplu simplu de folosire a informației obținute în timpul procesului de minimizare pentru a calcula aproximativ inversa matricei Hessian, metoda are anumite limitări și dezavantaje.

8.3 Metoda Davidon-Fletcher-Powell

Prima și cu siguranță una din cele mai bune scheme de construcție a inversei matricei Hessian a fost propusă de Davidon și îmbunătățită ulterior de Fletcher și Powell. Ea

are o proprietate uimitoare și anume că pentru o funcție obiectiv pătratică generează simultan direcțiile din metoda gradientilor conjugați și construiește inversa matricei Hessian. La fiecare pas aproximația inversei matricei Hessian este corectată prin intermediul a două matrice simetrice de rangul unu, și de aceea această schemă este adesea numită **procedura de corecție de rangul doi**. Acestei metode i se spune **metoda metricei variabile**, nume sugerat inițial de Davidon. Formula nou propusă pentru calculul aproximării inversei matricei Hessian este

$$\mathbf{S}_{k+1} = \mathbf{S}_k + \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{\mathbf{S}_k \mathbf{q}_k \mathbf{q}_k^T \mathbf{S}_k}{\mathbf{q}_k^T \mathbf{S}_k \mathbf{q}_k}. \quad (8.14)$$

Algoritmul metodei are următorii pseudocod:

1. Alege \mathbf{S}_0 simetrică și pozitiv definită

Alege \mathbf{x}_0

$k = 0$

Calculează $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$

2. **repetă**

2.1. $\mathbf{v}_k = -\mathbf{S}_k \mathbf{g}_k$

2.2. Minimizează $t(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{v}_k)$ în raport cu $\alpha \geq 0 \Rightarrow \alpha_k$

2.3. $\mathbf{p}_k = \alpha_k \mathbf{v}_k$

2.4. $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$

2.5. Calculează $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$

2.6. $\mathbf{q}_k = \mathbf{g}_{k+1} - \mathbf{q}_k$

2.7. $\mathbf{S}_{k+1} = \mathbf{S}_k + \mathbf{p}_k \mathbf{p}_k^T / (\mathbf{p}_k^T \mathbf{q}_k) - \mathbf{S}_k \mathbf{q}_k \mathbf{q}_k^T \mathbf{S}_k / (\mathbf{q}_k^T \mathbf{S}_k \mathbf{q}_k)$

2.8 $k = k + 1$

până când este îndeplinită condiția de oprire

Se poate demonstra că, dacă \mathbf{S}_k este pozitiv definită, atunci \mathbf{S}_{k+1} este și ea pozitiv definită. Este interesant că această afirmație este adevărată chiar dacă α_k nu este un punct de minim pentru funcția $t(\alpha)$.

Se poate demonstra de asemenea că, dacă f este o funcție pătratică, având deci o matrice Hessian constantă \mathbf{H} , atunci metoda Davidon-Fletcher-Powell generează direcții \mathbf{p}_k care sunt \mathbf{H} -ortogonale, iar după n pași $\mathbf{S}_n = \mathbf{H}^{-1}$. În acest caz, metoda face minimizări liniare succesive de-a lungul unor direcții conjugate. Mai mult, dacă aproximarea inițială \mathbf{S}_0 este luată matricea unitate, atunci metoda devine metoda gradientilor conjugați iar soluția se obține după n pași.

8.4 Clasa de metode Broyden

Formula pentru calculul matricei \mathbf{S}_{k+1} în cele două paragrafe anterioare se bazează pe relația

$$\mathbf{S}_{k+1}\mathbf{q}_i = \mathbf{p}_i, \quad 0 \leq i \leq k, \quad (8.15)$$

care a fost dedusă din

$$\mathbf{q}_i = \mathbf{H}\mathbf{p}_i, \quad 0 \leq i \leq k, \quad (8.16)$$

relație care este adevărată dacă funcția f este pătratică. O altă idee este de a folosi aproximații chiar ale matricei Hessian și nu ale inversei ei. Astfel, notând aproximațiile lui \mathbf{H} cu \mathbf{T}_k , vom căuta în mod analog să avem satisfăcute relațiile:

$$\mathbf{q}_i = \mathbf{T}_{k+1}\mathbf{p}_i, \quad 0 \leq i \leq k. \quad (8.17)$$

Relația (8.17) are aceeași formă ca și relația (8.15), numai că locul lui \mathbf{q}_i este luat de \mathbf{p}_i iar în loc de \mathbf{S} apare \mathbf{T} . Aceasta implică faptul că orice formulă pentru \mathbf{S} dedusă astfel încât relația (8.15) să fie satisfăcută poate fi transformată într-o formulă corespunzătoare pentru \mathbf{T} . Formula *complementară* se obține schimbând \mathbf{S} cu \mathbf{T} și interschimbând \mathbf{q} cu \mathbf{p} . Asemănător, orice formulă dedusă pentru \mathbf{T} astfel încât relația (8.17) să fie satisfăcută, poate fi transformată într-o formulă complementară pentru \mathbf{S} . Evident, complementul unui complement restaurează formula inițială.

De exemplu, formulele următoare realizează corecția de rangul unu:

$$\mathbf{S}_{k+1} = \mathbf{S}_k + \frac{(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)^T}{\mathbf{q}_k^T(\mathbf{p}_k - \mathbf{S}_k\mathbf{q}_k)}, \quad (8.18)$$

$$\mathbf{T}_{k+1} = \mathbf{T}_k + \frac{(\mathbf{q}_k - \mathbf{T}_k\mathbf{p}_k)(\mathbf{q}_k - \mathbf{T}_k\mathbf{p}_k)^T}{\mathbf{p}_k^T(\mathbf{q}_k - \mathbf{T}_k\mathbf{p}_k)}. \quad (8.19)$$

Similar, formula Davidon-Fletcher-Powell (sau, mai simplu, DFP)

$$\mathbf{S}_{k+1}^{\text{DFP}} = \mathbf{S}_k + \frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{S}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{S}_k}{\mathbf{q}_k^T\mathbf{S}_k\mathbf{q}_k} \quad (8.20)$$

are următoarea complementară:

$$\mathbf{T}_{k+1} = \mathbf{T}_k + \frac{\mathbf{q}_k\mathbf{q}_k^T}{\mathbf{q}_k^T\mathbf{p}_k} - \frac{\mathbf{T}_k\mathbf{p}_k\mathbf{p}_k^T\mathbf{T}_k}{\mathbf{p}_k^T\mathbf{T}_k\mathbf{p}_k} \quad (8.21)$$

Relația (8.21) se numește formula **Broyden-Fletcher-Goldfarb-Shanno** (sau mai simplu BFGS) pentru calculul aproximației \mathbf{T}_k a matricei Hessian și ea joacă un rol important în cele ce urmează.

O altă modalitate de a trece de la o formulă pentru \mathbf{S} la o formulă pentru \mathbf{T} este aceea de a calcula inversa. Această afirmație este evidentă, deoarece luând inversa relației (8.15) rezultă că:

$$\mathbf{q}_i = \mathbf{S}_{k+1}^{-1} \mathbf{p}_i, \quad 0 \leq i \leq k, \quad (8.22)$$

deci matricea \mathbf{S}_{k+1}^{-1} satisface relația (8.17). Se demonstrează că inversa unei formule de rangul doi este de asemenea o formula de rangul doi.

Noua formulă poate fi găsită explicit aplicând de două ori formula Sherman-Morrison:

$$(\mathbf{A} + \mathbf{a}\mathbf{b}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{a}\mathbf{b}^T\mathbf{A}^{-1}}{1 + \mathbf{b}^T\mathbf{A}^{-1}\mathbf{a}}, \quad (8.23)$$

unde \mathbf{A} este o matrice de dimensiuni $n \times n$, iar \mathbf{a} și \mathbf{b} sunt vectori coloană de dimensiune n .

Exercițiul 8.3: *Demonstrați formula Sherman-Morrison. Ce condiție trebuie să îndeplinească matricea \mathbf{A} ?*

Calculând inversa matricei \mathbf{T}_{k+1} dată de relația (8.21) rezultă că formula de calcul a aproximației inversei matricei Hessian în cazul metodei BFGS este

$$\mathbf{S}_{k+1}^{\text{BFGS}} = \mathbf{S}_k + \left(\frac{1 + \mathbf{q}_k^T \mathbf{S}_k \mathbf{q}_k}{\mathbf{q}_k^T \mathbf{p}_k} \right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{\mathbf{p}_k \mathbf{q}_k^T \mathbf{S}_k + \mathbf{S}_k \mathbf{q}_k \mathbf{p}_k^T}{\mathbf{q}_k^T \mathbf{p}_k}. \quad (8.24)$$

Algoritmul metodei BFGS este același cu cel al metodei DFP, numai că la pasul 2.7 al algoritmului se folosește formula (8.24). Experimentele numerice au arătat că performanța formulei BFGS este superioară celei DFP, și de aceea ea este preferată.

Se poate observa că atât formula DFP cât și formula BFGS folosesc o corecție de rangul doi care este construită cu ajutorul vectorilor \mathbf{p}_k și $\mathbf{S}_k \mathbf{q}_k$. Combinații ponderate ale acestor formule vor fi de aceea de același tip (simetrice, de rangul doi, și construite din \mathbf{p}_k și $\mathbf{S}_k \mathbf{q}_k$). Această observație a condus în mod natural la considerarea unei familii întregi de metode, cunoscute sub numele de **metode de tip Broyden**, definite de relația

$$\mathbf{S}^\Phi = (1 - \Phi) \mathbf{S}^{\text{DFP}} + \Phi \mathbf{S}^{\text{BFGS}}, \quad (8.25)$$

unde Φ este un parametru care poate lua orice valoare reală.

Exercițiul 8.4: *a) Ce semnificație au $\Phi = 0$ și, respectiv, $\Phi = 1$ în formula (8.25)?*

b) Studiați dacă clasa de metode Broyden include și formula care calculează \mathbf{S}_k folosind o corecție de rangul unu.

Făcând calcule algebrice care nu sunt extrem de simple, se poate arăta că relația (8.25) care definește clasa de metode Broyden poate fi scrisă explicit astfel:

$$\mathbf{S}_{k+1}^\Phi = \mathbf{S}_{k+1}^{\text{DFP}} + \Phi \mathbf{u}_k \mathbf{u}_k^T, \quad (8.26)$$

unde

$$\mathbf{u}_k = \sqrt{\mathbf{q}_k^T \mathbf{S}_k \mathbf{q}_k} \left(\frac{\mathbf{p}_k}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{\mathbf{S}_k \mathbf{q}_k}{\mathbf{q}_k^T \mathbf{S}_k \mathbf{q}_k} \right). \quad (8.27)$$

Observații:

1. O metodă Broyden este definită ca fiind o metodă quasi-Newton în care, la fiecare iterație, se folosește o formulă de tip Broyden (8.25) pentru calculul aproximației inversei matricei Hessian. În general, parametrul Φ poate varia de la o iterație la alta, deci o metodă Broyden este caracterizată de un șir Φ_1, Φ_2, \dots . Despre o metodă Broyden se spune că este *pură* dacă folosește o singură valoare pentru Φ .
2. Deoarece atât \mathbf{S}^{DFP} cât și \mathbf{S}^{BFGS} satisfac relația fundamentală (8.15) rezultă că această relație este satisfăcută de orice metodă din clasa Broyden.
3. O metodă Broyden nu asigură în mod necesar faptul că \mathbf{S}^Φ este pozitiv definită pentru orice valoare a lui Φ . Deoarece în metoda DFP pozitiv definirea matricei \mathbf{S} este asigurată, rezultă din relația (8.26) că pozitiv definirea este păstrată pentru orice $\Phi \geq 0$ deoarece suma dintre o matrice pozitiv definită și o matrice pozitiv semidefinită este pozitiv definită. Pentru $\Phi < 0$ există posibilitatea ca \mathbf{S}^Φ să devină singulară și de aceea, în acest caz, trebuie luate anumite precauții. În practică se folosește de obicei $\Phi \geq 0$ pentru a evita dificultățile.
4. Pentru a determina care sunt strategiile potrivite pentru selecția parametrilor Φ_k au fost făcute multe studii și experimente numerice cu metodele Broyden. S-a demonstrat că alegerea acestor parametri nu este relevantă în cazul unor funcții pătratice și a unor minimizări liniare făcute cu precizie. În mod surprinzător, s-a arătat că, și în cazul funcțiilor care nu sunt pătratice, dacă minimizările liniare sunt făcute cu precizie, punctele generate de orice metodă Broyden sunt identice. Concluzia este că diferențele dintre metode sunt importante atunci când minimizările liniare nu sunt făcute cu acuratețe.

Exercițiul 8.5: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere corespunzătoare acestei teme.

b) Programul **main_quasiNewton.sci** minimizează funcția pătratică (7.20) prin metode de tip quasi-Newton. Programul permite alegerea uneia din metodele: corecție de rangul unu, Davidon-Fletcher-Powell, sau o metodă din clasa Broyden (în acest caz trebuie introdusă valoarea lui Φ). Cum se face minimizarea liniară a funcției?

c) Implementați o variantă de program în care pentru minimizare să se folosească metoda secțiunii de aur.

d) Implementați o variantă de program în care să se minimizeze funcția dată de relația (7.31) folosind un calcul exact pentru minimizare.

e) Implementați o variantă de program în care să se minimizeze funcția dată de relația (7.31) folosind un calcul aproximativ pentru minimizare.

f) Implementați o variantă de program în care să se minimizeze funcția lui Rosenbrock (“banană”) definită la capitolul 5.

Exercițiul 8.6: Să se studieze comportarea diferitelor metode quasi-Newton în cazul următoarelor funcții:

- funcția pătratică definită de expresia (7.20). Pentru minimizarea liniară se va folosi calculul exact, apoi calculul aproximativ cu metoda secțiunii de aur. În acest din urmă caz se va studia influența parametrilor metodei secțiunii de aur.
- funcția pătratică de n variabile definită de relația (7.31). Pentru minimizarea liniară se va folosi calculul exact, apoi calculul aproximativ cu metoda secțiunii de aur. În acest din urmă caz se va studia influența parametrilor metodei secțiunii de aur. Se va studia influența lui n asupra efortului de calcul.
- funcția lui Rosenbrock definită la capitolul 5.

Exercițiul 8.7: Comparați rezultatele studiului de la exercițiul 8.6 cu rezultatele obținute la metoda gradientului și metoda gradientilor conjugați.

8.5 Metode de metrică variabilă sau gradienti conjugați?

Există foarte multe asemănări între metodele de metrică variabilă și metoda gradientilor conjugați³. Ambele metode au la bază ideea de a folosi informația obținută din minimizări unidimensionale de-a lungul unor direcții succesive, algoritmi fiind construiți astfel încât n minimizări liniare să conducă către minimul exact al unei funcții pătratice în n dimensiuni. Pentru funcții mai generale, nepătratice, ambele metode oferă o combinație de avantaje care le fac atractive. Printre acestea, direcțiile generate sunt întotdeauna coborâtoare, iar după n iterații convergența este superliniară.

Metodele de metrică variabilă diferă de metoda gradientilor conjugați prin faptul că memorează și reactualizează informația acumulată. În loc să memoreze un vector

³De altfel, metoda gradientilor conjugați este un caz particular al metodelor de metrică variabilă.

intermediar de dimensiune n , ele memorează o matrice de dimensiune $n \times n$. În general, pentru n moderat, acesta nu este un dezavantaj semnificativ.

Dezvoltate mai devreme și utilizate mai mult, metodele de metrică variabilă au mai mulți utilizatori care le acordă încredere. În plus, există multe implementări sofisticate ale metodelor de metrică variabilă, care minimizează eroarea de rotunjire sau tratează condiții mai speciale.

Exercițiul 8.8: Dacă în viitor va trebui să folosiți o metodă deterministă de ordinul unu pentru a găsi minimumul unei funcții, ce metodă veți alege? Motivați alegerea făcută, în funcție de dimensiunea n a problemei.

Capitolul 9

Minimizări multidimensionale - metode stocastice de optimizare. Algoritmi genetici.

În cadrul acestui capitol se prezintă câteva dintre metodele stocastice de căutare a unui optim.

Reamintim că metodele de optimizare pot fi clasificate în două mari categorii: metode deterministe și metode stocastice.

- **Metodele deterministe** conduc la aceeași soluție pentru rulări diferite ale programului dacă pornesc de la aceleași condiții inițiale și au aceleași parametri.
- **Metodele stocastice** au un caracter aleator și ele nu conduc în mod necesar la aceeași soluție, chiar dacă algoritmul pornește din aceleași condiții inițiale și are aceleași parametri.

Dezavantajul metodelor deterministe este acela că ele găsesc întotdeauna un extrem local, dependent de inițializare. De cele mai multe ori se dorește însă găsirea unui extrem global, lucru ce pretinde explorarea întregului domeniu de căutare și nu numai o vecinătate a inițializării. Pentru a determina un extrem global se poate proceda astfel: se execută algoritmul determinist pentru mai multe puncte inițiale de căutare împrăștiate uniform în domeniul de căutare și apoi se alege dintre soluțiile găsite valoarea cea mai bună sau se perturbă un extrem local găsit pentru a vedea dacă algoritmul determinist cu această inițializare regăsește același extrem. Ca o alternativă la aceste două abordări, au început să fie folosiți tot mai des algoritmi stocastici. Aceștia nu garantează găsirea unui extrem global, dar ei au o probabilitate mult mai mare de a găsi un astfel de extrem. De asemenea ei mai au avantajul că nu necesită evaluarea derivatelor funcției de optimizat,

fiind în consecință algoritmi de ordinul zero. Dezavantajul lor este acela că numărul de evaluări de funcții necesar pentru găsirea optimului este relativ mare față de cazul metodelor deterministe, dar în multe situații acest sacrificiu trebuie făcut, pentru că acești algoritmi sunt singurii care dau rezultate numerice acceptabile.

9.1 Metoda căutării aleatoare (drumului aleator)

9.1.1 Varianta Matyas

Metoda căutării aleatoare¹ explorează domeniul variabilelor funcției obiectiv într-un mod aleator. Iată câteva avantaje ale metodei: necesită doar evaluarea funcției nu și a derivatei, este simplă și ușor de înțeles, este ușor de adaptat unei anumite aplicații. Mai mult, s-a demonstrat că această metodă converge către un optim global pe o mulțime compactă.

Fie $f(\mathbf{x})$ funcția obiectiv de minimizat. Prima variantă a acestei metode a fost propusă de Matyas și poate fi descrisă în următorii patru pași:

- **Pasul 1:** Alege un punct de start \mathbf{x} din domeniul de definiție.
- **Pasul 2:** Adaugă un vector aleator \mathbf{dx} punctului curent \mathbf{x} din spațiul de căutare și evaluează funcția obiectiv în noul punct curent $\mathbf{x} + \mathbf{dx}$.
- **Pasul 3:** Dacă $f(\mathbf{x} + \mathbf{dx}) < f(\mathbf{x})$ atunci noul punct curent de căutare este $\mathbf{x} + \mathbf{dx}$.
- **Pasul 4:** Algoritmul se oprește dacă a fost atins un anumit număr de evaluări de funcții obiectiv, altfel, algoritmul se reia de la pasul 2.

Exercițiul 9.1: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere corespunzătoare metodei căutării aleatoare. Acestea constituie implementarea acestei metode pentru determinarea minimului funcției “cămila”. (Definiția acestei funcții se găsește la capitolul 5.)

b) Descrieți conținutul acestor fișiere.

c) Executați de 10 ori programul `main_random_search.sci`. Contorizați numărul de eșecuri (de câte ori minimul găsit nu este în zona minimului global). În situațiile în care zona de minim global a fost găsită evaluați eroarea soluției și observați numărul de puncte curente de căutare.

Se observă caracterul aleator al metodei, în sensul că direcțiile de căutare sunt determinate de un generator de numere aleatoare. Figura 9.1 prezintă un exemplu de traiectorie a punctului de căutare în această metodă.

¹ “Random search”

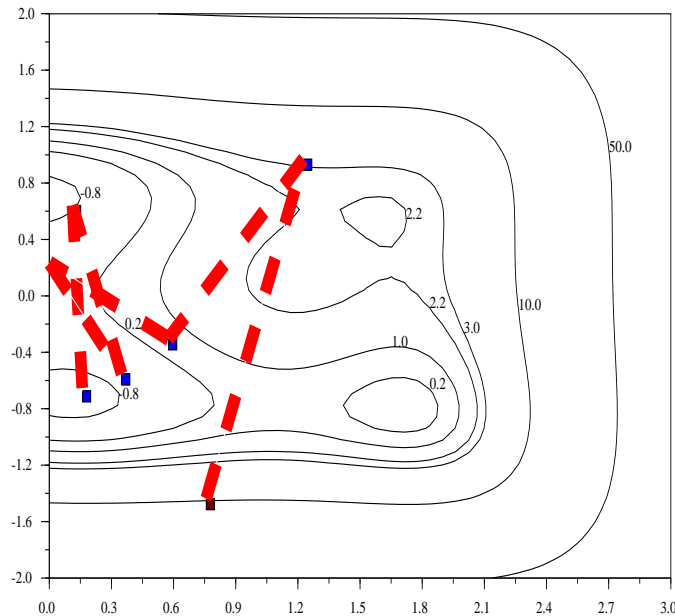


Figura 9.1: Exemplu de căutare în metoda random search - varianta Matyas.

Exercițiul 9.2: Modificați programul principal astfel încât să se minimizeze funcția lui Rosenbrock. Pentru 10 rulări independente evaluați eroarea.

Exercițiul 9.3: Studiați influența condiției de oprire asupra soluției numerice.

Exercițiul 9.4: Reprezentați grafic variația erorii relative a soluției numerice în funcție de numărul de evaluări, pentru un caz de succes.

Exercițiul 9.5: Ce efect ar avea micșorarea lungimii vectorului \mathbf{dx} în timpul căutării?

9.1.2 Varianta îmbunătățită

Există câteva modalități de a îmbunătăți această versiune primitivă a metodei, bazate pe următoarele observații:

- Dacă alegerea unei anumite direcții duce către o funcție obiectiv mai mare atunci direcția opusă ar putea duce către o funcție obiectiv mai mică.
- Dacă o anumită direcție a căutărilor succesive s-a dovedit de succes, ea ar trebui să polarizeze căutărilor ulterioare. Pe de altă parte eșecurile succesive într-o anumită

direcție ar trebui să descurajeze căutările ulterioare în acea direcție.

Folosind aceste două observații, o versiune modificată a metodei căutării aleatoare ar putea fi descrisă în următorii șase pași:

- **Pasul 1:** Alege un punct de start inițial \mathbf{x} drept punct curent și setează “polarizarea” inițială \mathbf{p} egală cu un vector nul.
- **Pasul 2:** Adaugă un termen \mathbf{p} și un vector aleator \mathbf{dx} punctului curent \mathbf{x} și evaluează funcția obiectiv în punctul $\mathbf{x} + \mathbf{p} + \mathbf{dx}$.
- **Pasul 3:** Dacă $f(\mathbf{x} + \mathbf{p} + \mathbf{dx}) < f(\mathbf{x})$ atunci noul punct curent este $\mathbf{x} + \mathbf{p} + \mathbf{dx}$ iar noul termen de polarizare este $0.2\mathbf{p} + 0.4\mathbf{dx}$ și sări la pasul 6, altfel mergi la pasul următor.
- **Pasul 4:** Dacă $f(\mathbf{x} + \mathbf{p} - \mathbf{dx}) < f(\mathbf{x})$ atunci noul punct curent este $\mathbf{x} + \mathbf{p} - \mathbf{dx}$ iar noul termen de polarizare este $\mathbf{p} - 0.4\mathbf{dx}$ și sări la pasul 6, altfel mergi la pasul următor.
- **Pasul 5:** Setează noua polarizare $0.5\mathbf{p}$ și mergi la pasul 6.
- **Pasul 6:** Stop dacă a fost atins numărul maxim de evaluări de funcții obiectiv. Altfel reia de la pasul 2.

O îmbunătățire suplimentară a algoritmului ar putea fi obținută dacă componentele vectorului \mathbf{dx} ar scădea în timp.

Exercițiul 9.6: a) *Ce dificultăți apar în cazul în care funcția de minimizat are restricții de domeniu?*

b) *Implementați (opțional) varianta îmbunătățită a metodei căutării aleatoare. Studiați influența parametrilor algoritmului (valorile coeficienților folosiți în calculul polarizării) asupra performanțelor sale în diferite cazuri concrete.*

Observație: metoda căutării aleatoare este o metodă de optimizare dedicată în principal optimizării problemelor în care parametrii au variație continuă. Se poate folosi această metodă și în cazul în care parametrii de optimizat au variație discretă dar atunci observațiile precedente s-ar putea să nu fie adevărate și algoritmul Matyas s-ar putea să fie mai potrivit.

9.2 Programe evoluționiste. Algoritmi genetici.

Programele evoluționiste (în particular algoritmi genetici) sunt metode de optimizare aleatoare bazate pe modele biologice evoluționiste. Ei au fost propuși și investigați de

John Holland în 1975. Spre deosebire de metodele de căutare aleatoare, aceste programe operează nu cu o singură soluție numerică ci cu o familie de soluții, împrăștiate în spațiul de căutare și care evoluează spre soluția problemei (se îngrămădesc progresiv către aceasta).

9.2.1 Structura unui program evoluționist

Un program de evoluție este un algoritm probabilistic care menține o **populație** $P(t)$ definită la iterația t ca o mulțime de **indivizi** x_k^t :

$$P(t) = \{x_1^t, x_2^t, \dots, x_n^t\}. \quad (9.1)$$

Un individ (numit și **genotip** sau **cromozom**) este o soluție potențială a problemei, implementată ca o structură de date \mathcal{S} . Folosind funcția obiectiv, oricărui individ, prin evaluare i se atribuie un **grad de adecvare** (*fitness measure*). Ideea comună oricărui program bazat pe evoluție este aceea că o populație de indivizi suferă transformări și în timpul acestui proces indivizii luptă pentru supraviețuire iar populația tinde către un grad de adecvare cât mai mare.

La fiecare iterație nouă ($t+1$) se formează o populație (**generație**) nouă prin **selecția** indivizilor cu un grad de adecvare mai bun.

Membrii populației suferă transformări, numite de **alterare**, modificările făcându-se cu ajutorul **operatorilor genetici**. Aceștia sunt de două tipuri:

- unari, de tip **mutație**: un astfel de operator m crează un individ nou printr-o schimbare a unui individ vechi:

$$m : \mathcal{S} \rightarrow \mathcal{S};$$

- binari, de tip **încrucișare** (*crossover*): un astfel de operator c crează un individ prin combinarea a doi indivizi vechi:

$$c : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}.$$

După un anumit număr de generații cel mai bun individ (cel mai apropiat de optim) este adoptat ca soluția numerică a problemei de optimizare.

Iată **algoritmul unui program de evoluție**:

1. $t = 0$
2. inițializează $P(t)$

3. evaluează $P(t)$

4. **cât timp**(nu condiție de stop) **repetă**

4.1. $t = t + 1$

4.2. selectează $P(t)$ din $P(t - 1)$

4.3. alterează $P(t)$

4.4. evaluează $P(t)$

Pentru o anumită problemă pot fi formulate mai multe programe de evoluție. Ele pot diferi din mai multe puncte de vedere:

- structura de date folosită pentru reprezentarea unui individ (\mathcal{S});
- metoda de creare a populației inițiale;
- operatorii genetici și de selecție utilizați;
- parametrii algoritmului: dimensiunea populației, probabilitatea aplicării diferiților operatori;
- metoda de tratare a restricțiilor;
- criteriul de oprire folosit.

Structura unui algoritm genetic este aceeași ca cea prezentată pentru un program de evoluție. Diferențele sunt la nivele mai joase. Algoritmii genetici clasici folosesc drept structură de date pentru reprezentarea unui individ un șir binar de lungime fixă și doi operatori: mutația binară și încrucișarea binară. În algoritmii evoluționiști cromozomii pot fi reprezentați și altfel decât prin structuri binare de lungime fixă, iar procesul de alterare poate include și alți operatori genetici.

Un algoritm genetic, ca orice program de evoluție, are următoarele cinci componente:

1. reprezentarea genetică pentru soluția potențială a problemei;
2. o modalitate de a crea o populație inițială a soluției potențiale;
3. o funcție de evaluare care este funcția de cost ce trebuie minimizată și care joacă rolul mediului;
4. operatori genetici care alterează compoziția populației;
5. valori pentru parametri: dimensiunea populației, probabilitățile cu care sunt aplicați diferiți operatori.

9.2.2 Algoritmi genetici

Conceptele de bază ale unui algoritm genetic sunt prezentate în cele ce urmează.

Reprezentarea genetică (schema de codificare). Aceasta transformă punctele din spațiul de căutare în șiruri binare. De exemplu, un punct $(11, 6, 9)$ într-un spațiu tridimensional poate fi reprezentat ca trei șiruri binare concatenate, care reprezintă cromozomul individului:

$$\underbrace{1011}_{11} \underbrace{0110}_6 \underbrace{1001}_9$$

în care fiecare parametru este codificat ca o **genă** compusă din patru biți, folosind scrierea binară. Există și alte metode de codificare (de exemplu codificarea Gray). Schema de codificare joacă un rol hotărâtor în determinarea performanțelor unui algoritm genetic. Mai mult, operatorii genetici ca mutația și încrucișarea trebuie să fie adecvați schemei de reprezentare folosite.

Evaluarea gradului de adecvare. Primul pas după crearea unei generații este de a calcula gradul de adecvare a fiecărui membru din populație. Pentru o problemă de minimizare (maximizare), gradul de adecvare f_i al fiecărui membru reprezintă de obicei funcția obiectiv evaluată pentru acel individ (punct). De obicei sunt utilizate scalări sau translații ale valorilor funcției (de exemplu pentru a obține numai valori pozitive ale funcției). O altă abordare este de a utiliza poziția indivizilor (după sortarea lor) în populație. Această abordare are avantajul că nu este necesară evaluarea cu acuratețe a funcției atâta vreme cât se păstrează ierarhia indivizilor.

Mecanismul de selecție. După evaluare, trebuie creată o populație nouă din generația curentă. Mecanismul de selecție stabilește care indivizi (**părinți**) vor produce indivizi noi (**copii**) în generația următoare și este analogul conceptului de *supraviețuirea celui mai bun* din selecția naturală. De obicei probabilitatea ca un individ să fie selectat pentru încrucișare este proporțională cu gradul său de adecvare. Cel mai des întâlnit mod este de a folosi probabilitatea de selecție egală cu $f_i / \sum_{k=1}^n f_k$, unde n este dimensiunea populației. Efectul acestei metode de selecție este de a favoriza să devină părinți acei membri care au grade de adecvare peste medie.

Încrucișarea. Pentru a exploata potențialul unei generații se folosește operatorul de încrucișare. Copiii rezultați după încrucișare vor reține (se speră) trăsăturile bune ale părinților. Încrucișarea este aplicată prin selecția unei perechi de părinți cu o probabilitate egală cu **probabilitatea (rata) de încrucișare**. Cel mai des întâlnită este *încrucișarea într-un singur punct*, în care se selectează aleator un punct de încrucișare² și cei doi părinți schimbă între ei biții ce urmea punctului (figura 9.2). În cazul *încrucișării în două puncte*

²Punctul de încrucișare reprezintă poziția de încrucișare în genele șirului binar al indivizilor.

există două puncte de încrucișare și părinții schimbă între ei șirul de biți dintre aceste două puncte (figura 9.3). Există și alte metode de încrucișare.

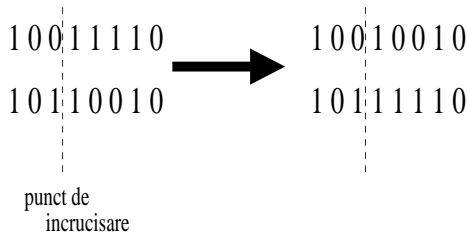


Figura 9.2: Încrucișare într-un singur punct

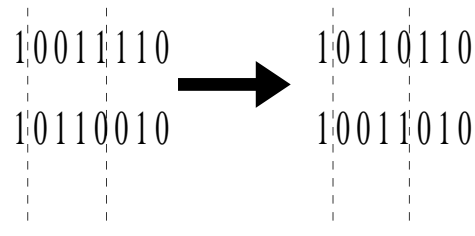


Figura 9.3: Încrucișare în două puncte

Efectul încrucișării este similar celui din procesul evoluției naturale, în care părinții dau copiilor părți din proprii lor cromozomi. De aceea, unii copii sunt capabili să-și depășească părinții în performanțe dacă au primit gene bune de la părinții lor.

Mutația. Încrucișarea exploatează potențialul unei generații, dar este posibil ca populația să nu conțină toată informația necesară rezolvării unei anumite probleme. Din acest motiv, se folosește un operator de mutație capabil să genereze noi cromozomi. Cea mai obișnuită metodă de a implementa mutația este de a modifica un bit (din 1 în 0 sau invers) ales cu o probabilitate egală cu **probabilitatea (rata) de mutație**, aceasta fiind un număr cu valoare mică (figura 9.4).

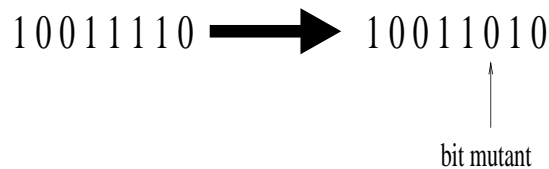


Figura 9.4: Mutația

Efectul mutației este foarte important pentru că ea previne ca populația să convergă către un extrem local. Rata mutației este de obicei mică pentru a nu conduce la pierderea cromozomilor buni. Dacă rata mutației este mare (mai mare decât 0.1), algoritmul genetic se comportă mai degrabă ca un algoritm de căutare aleatoare.

Acest paragraf dă doar o descriere generală a trasăturilor de bază ale algoritmilor genetici. Detaliile de implementare pot fi foarte variate. De exemplu se practică de multe ori o politică de a păstra cei mai buni indivizi în noua generație. Acest principiu poartă numele de **elitism**.

Vom prezenta acum cel mai simplu exemplu de algoritm genetic.

Enunțul problemei

Să ne imaginăm că vrem să găsim minimul funcției *cămila* $C(x, y)$ în domeniul $[0.2] \times [-1.2, 1]$ și că avem la dispoziție un program care implementează un algoritm genetic dedicat maximizării funcțiilor pozitive. În consecință, vom scala funcție $C(x, y)$. Astfel, în loc să minimizăm $C(x, y)$ vom maximiza funcția

$$F(x, y) = 3 - C(x, y). \quad (9.2)$$

Stabilirea schemei de reprezentare

Dorim să optimizăm funcția F cu o anumită precizie, să presupunem n cifre semnificative exacte. Pentru a atinge o asemenea precizie, domeniul unei variabile (să presupunem că notăm acest domeniu cu $[a_i, b_i]$ unde a_i reprezintă limita inferioară și b_i reprezintă limita superioară a domeniului variabilei i) trebuie împărțit în $(b_i - a_i)10^n$ intervale egale. Dacă notăm cu m_i cel mai mic număr întreg pentru care $(b_i - a_i)10^n \leq 2^{m_i} - 1$, atunci o reprezentare binară având fiecare parametru m_i calculat astfel va satisface această cerință de precizie. Lungimea cromozomului ce va codifica un individ este $m = \sum_i m_i$.

Pentru exemplul considerat, pentru a obține o cifră semnificativă după virgulă trebuie făcute următoarele raționamente:

- pentru variabila x , lungimea domeniului ei este 2, numărul de intervale este 20, și deoarece $2^4 < 20 \leq 2^5 - 1$, rezultă că sunt necesar 5 biți pentru codificare.
- variabila y are lungimea domeniului 2.2, numărul de intervale este 22 și numărul de biți necesari este de asemenea 5.

În consecință un cromozom va avea 10 biți.

Exercițiul 9.7: Cum va trebui făcută codificarea variabilelor acestei probleme pentru a obține o precizie de 6 cifre semnificative?

Decodificarea unei variabile (conversia numărului binar în valoarea reală a variabilei) se face identificând mai întâi șirurile binare ce codifică fiecare variabilă (se identifică genele), apoi, pentru fiecare astfel de șir binar se calculează parametrul x_i astfel:

- se convertește șirul binar din număr în baza 2 în număr în baza 10 (să notăm acest număr cu x'_i ;
- se calculează numărul real x_i :

$$x_i = a_i + x'_i \frac{b_i - a_i}{2^{m_i} - 1}. \quad (9.3)$$

Exercițiul 9.8: Verificați că individul (corespunzător exemplului din acest paragraf) a cărui codificare este 1101101101 se decodifică în $(1.7419355, -0.2774194)$.

Populația inițială

Pentru a inițializa populația, fiecare bit este ales în mod aleator 1 sau 0. Să presupunem că populația are 7 indivizi și iată un exemplu de populație inițială:

0.	1.	0.	0.	1.	0.	1.	0.	1.	0.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
1.	1.	1.	0.	0.	1.	1.	0.	1.	0.
1.	0.	1.	1.	0.	1.	1.	0.	1.	0.
0.	0.	1.	1.	1.	1.	0.	1.	0.	1.
1.	1.	0.	0.	1.	0.	0.	1.	0.	1.
0.	1.	1.	1.	0.	1.	0.	1.	0.	1.

Această populație, decodificată reprezintă punctele:

.580645161	-.490322581
.193548387	-1.05806452
1.80645161	.64516129
1.41935484	.64516129
.451612903	.290322581
1.61290323	-.84516129
.903225806	.290322581

pentru care funcția obiectiv are următoarele valori:

2.89250436
2.52274865
.532711696
.795400409
2.44632759
3.11703488
.999916289

Cel mai bun individ este penultimul: 1100100101 adică $(1.61290323, -0.84516129)$, de valoare 3.117035.

Mecanismul de selecție

Procesul de selecție are caracter aleator și trebuie să acorde șanse mai mari indivizilor mai adecvați. Aceste caracteristici se obțin considerând o distribuție de probabilitate dată

de o ruletă care are feliile de dimensiuni proporționale cu valorile indivizilor. O astfel de roată de ruletă se construiește astfel:

- Se calculează valorile (gradele de adecvare) ale fiecărui cromozom. Să notăm aceste valori cu f_i ($i = 1, \text{pop_size}$, unde cu pop_size am notat dimensiunea populației).
- Se însumează aceste valori $f_{\text{total}} = \sum_{i=1}^{\text{pop_size}} f_i$.
- Se calculează probabilitățile de selecție p_i pentru fiecare cromozom:

$$p_i = \frac{f_i}{f_{\text{total}}}. \quad (9.4)$$

- Se calculează probabilitățile cumulate q_i pentru fiecare individ:

$$q_i = \sum_{j=1}^i p_j. \quad (9.5)$$

Procesul de selecție se bazează pe învârtirea roții de pop_size ori. La fiecare rotație se selectează un singur individ astfel:

- Se generează un număr aleator r în intervalul $[0, 1]$.
- Dacă $r < q_1$ atunci se selectează primul cromozom pentru a deveni părinte; altfel se selectează individul cu indice i ($2 \leq i \leq \text{pop_size}$) pentru care $q_{i-1} < r \leq q_i$.

Evident unii cromozomi pot fi selectați mai mult de o dată, fiind favorizați³ pentru a deveni părinți cei mai buni indivizi.

Pentru exemplu de mai sus, să presupunem că au fost selectați indivizii 5, 2, 2, 6, 4, 4, 7 și rezultă că populația după selecție va fi:

0.	0.	1.	1.	1.	1.	0.	1.	0.	1.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
1.	1.	0.	0.	1.	0.	0.	1.	0.	1.
1.	0.	1.	1.	0.	1.	1.	0.	1.	0.
1.	0.	1.	1.	0.	1.	1.	0.	1.	0.
0.	1.	1.	1.	0.	1.	0.	1.	0.	1.

³Indivizilor cu un grad de adecvare mai bun le corespunde pe roata de ruletă un sector de cerc mai mare.

Exercițiul 9.9: *Mecanismul de selecție de mai sus este valabil pentru maximizarea funcțiilor cu valori pozitive. Cum ar trebui modificat în cazul minimizării funcțiilor cu valori pozitive?*

Încrucișarea

După selecție se aplică operatorul de încrucișare. Unul din parametrii algoritmului genetic este probabilitatea de încrucișare p_c . Această probabilitate permite calculul numărului probabil de cromozomi care vor suferi încrucișarea, $p_c \cdot \text{pop_size}$. Se procedează astfel. Pentru fiecare cromozom:

- Se generează un număr real aleator r în intervalul $[0, 1]$.
- Dacă $r < p_c$ atunci cromozomul respectiv este ales pentru încrucișare.

Dacă numărul de cromozomi aleși pentru încrucișare este impar atunci se adaugă sau se elimină (în mod aleator) un cromozom.

Să presupunem în exemplul de mai sus că au fost aleși pentru încrucișare indivizii 5 și 7 și aceștia au fost încrucișați la poziția 7 (poziție care este aleasă aleator), rezultă după încrucișare populația:

0.	0.	1.	1.	1.	1.	0.	1.	0.	1.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
1.	1.	0.	0.	1.	0.	0.	1.	0.	1.
0.	1.	1.	1.	0.	1.	0.	0.	1.	0.
1.	0.	1.	1.	0.	1.	1.	0.	1.	0.
1.	0.	1.	1.	0.	1.	1.	1.	0.	1.

Mutație

Următorul operator, mutația, acționează asupra biților în mod individual. Un alt parametru al algoritmului genetic este probabilitatea de mutație p_m . Acesta dă numărul așteptat de biți care suferă mutația și anume $p_m \cdot m \cdot \text{pop_size}$. Fiecare bit (din toți cromozomii populației) are o șansă egală de a suferi mutația adică schimbarea din 0 în 1 și viceversa. Se procedează astfel. Pentru fiecare cromozom din populația curentă (adică după încrucișare) și pentru fiecare bit din fiecare cromozom:

- Se generează un număr real aleator r în intervalul $[0, 1]$.
- Dacă $r < p_m$ bitul respectiv suferă mutația.

Aplicând această schemă exemplului de mai sus au rezultat următoarele mutații: individul 1 \rightarrow biții 1 și 3, individul 4 \rightarrow bitul 10, individul 5 \rightarrow bitul 3, iar populația după mutație este:

1.	0.	0.	1.	1.	1.	0.	1.	0.	1.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
0.	0.	0.	1.	1.	0.	0.	0.	1.	0.
1.	1.	0.	0.	1.	0.	0.	1.	0.	0.
0.	1.	0.	1.	0.	1.	0.	0.	1.	0.
1.	0.	1.	1.	0.	1.	1.	0.	1.	0.
1.	0.	1.	1.	0.	1.	1.	1.	0.	1.

După selecție, încrucișare și mutație, noua populație este gata pentru a fi reevaluată. Evaluarea este folosită pentru a construi noile funcții de probabilitate necesare procesului de selecție (pentru a construi o nouă roată de ruletă cu sectoarele ajustate după noile grade de adecvare). Restul evoluției este doar o repetare ciclică a pașilor de mai sus.

Figurile 9.5 și 9.6 reprezintă un exemplu de evoluție: figura 9.5 indică distribuția populației inițiale (20 de indivizi) iar figura 9.6 distribuția populației după 20 de generații.

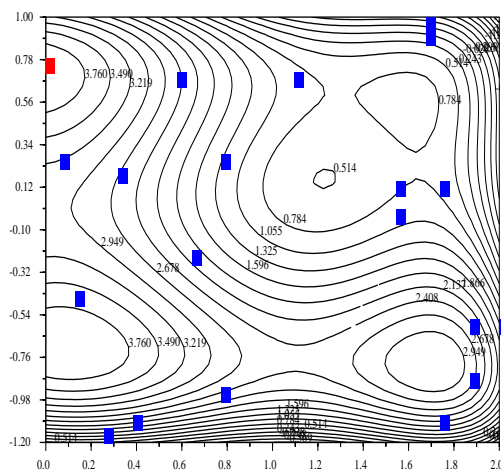


Figura 9.5: Populația inițială

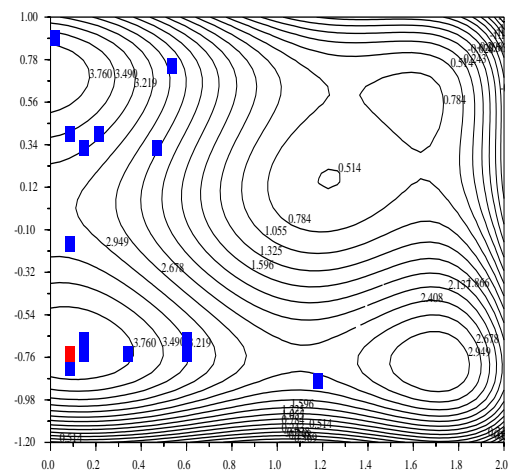


Figura 9.6: Populația după 20 generații

Exercițiul 9.10: a) Copiați de la adresa <http://www.lmn.pub.ro/~gabriela> arhiva de fișiere corespunzătoare acestei metode.

b) Descrieți conținutul acestor fișiere.

c) Descrieți problema de optimizat.

Exercițiul 9.11: Descrieți modul de implementare a componentei elitiste.

Exercițiul 9.12: Efectuați teste numerice pentru a putea observa influența parametrilor algoritmului genetic (dimensiunea populației, probabilitatea de încrucișare și de mutație, condiția de oprire) asupra convergenței algoritmului genetic. Observație: pentru fiecare set de parametri faceți câte 10 execuții independente pentru a observa comportarea medie la un set de parametri. Contorizați numărul de eșecuri (convergență în extreme locale) pentru fiecare test.

Exercițiul 9.13: Modificați programul Scilab astfel încât să se minimizeze funcția lui Rosenbrock.

Observație: Având în vedere că această funcție nu are extreme locale, nu vom putea vorbi de eșecuri, și vom putea evalua de fiecare dată eroarea față de extremul acestei funcții.

Exercițiul 9.14: a) Modificați programul Scilab astfel încât să se contorizeze numărul de evaluări de funcții obiectiv.

b) Pentru funcția Rosenbrock evaluați eroarea în funcție de numărul de evaluări de funcții. Comparați această evoluție cu graficul similar obținut cu metoda gradientilor conjugați (capitolul 7).

Exercițiul 9.15: Rezultatele testelor efectuate prezentați-le și grafic, sub forma variației celui mai bun individ (valoarea lui, parametrii lui) în funcție de numărul de evaluări de funcții, respectiv de indicele generației curente.

Capitolul 10

Aplicație la minimizările multidimensionale. Studiul bobinelor lui Helmholtz.

Aceast capitol urmărește aplicarea metodelor de optimizare prezentate în capitolele 5, 6 și 7 pentru optimizarea dispozitivului de producere a câmpului magnetic uniform, cunoscut sub numele de *bobinele lui Helmholtz*¹.

10.1 Bobinele Helmholtz

10.1.1 Descrierea dispozitivului

Dispozitivul cunoscut sub numele de *bobinele Helmholtz* este format din două bobine identice, coaxiale. Un astfel de dispozitiv crează în anumite condiții un câmp magnetic aproximativ uniform cel puțin într-o mică porțiune situată la mijlocul distanței dintre bobine. Pentru simplitate vom presupune că bobinele sunt de fapt spire (figura 10.1) coaxiale, paralele, parcurse de curenți egali, în același sens.

10.1.2 Considerații teoretice

Fie P un punct situat pe axa spirelor la distanța x' de mijlocul O al segmentului ce unește centrele spirelor (figura 10.1). Atunci câmpul magnetic produs în P de doi curenți egali

¹Temă pentru studenți – Concepeți un referat scris (se recomandă în Latex) care să reflecte modul în care s-au comportat metodele de optimizare studiate în rezolvarea acestei probleme. Vă recomandăm ca acest referat să fie bazat pe răspunsurile la exercițiile și întrebările din acest capitol.

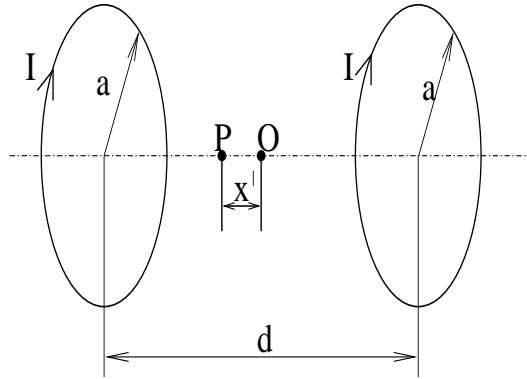


Figura 10.1: Bobinele Helmholtz.

și în același sens I ce străbat spirele are modulul

$$B(x') = \frac{\mu_0 I a^2}{2} \left\{ \left[\left(\frac{d}{2} + x' \right)^2 + a^2 \right]^{-3/2} + \left[\left(\frac{d}{2} - x' \right)^2 + a^2 \right]^{-3/2} \right\}. \quad (10.1)$$

Exercițiul 10.1: Demonstrați formula (10.1). Care este direcția și sensul câmpului magnetic în punctul P ?

Dacă notăm variabilele:

$$x = \frac{x'}{a}, \quad (10.2)$$

$$\beta = \frac{d}{2a}, \quad (10.3)$$

$$B_0 = \frac{\mu_0 I}{2a} \quad (10.4)$$

și funcțiile:

$$f_1(x) = [1 + (\beta + x)^2]^{-3/2}, \quad (10.5)$$

$$f_2(x) = [1 + (\beta - x)^2]^{-3/2}, \quad (10.6)$$

$$F(x) = f_1(x) + f_2(x), \quad (10.7)$$

atunci câmpul magnetic în P se poate scrie ca fiind

$$B(x) = B_0 F(x). \quad (10.8)$$

Exercițiul 10.2: Verificați formula (10.8).

Vom dezvoltă funcțiile f_1 și f_2 în serie Taylor în jurul lui $x = 0$:

$$f_1(x) = f_1(0) + xf'_1(0) + \frac{x^2}{2!}f''_1(0) + \dots \quad (10.9)$$

$$f_2(x) = f_2(0) + xf'_2(0) + \frac{x^2}{2!}f''_2(0) + \dots \quad (10.10)$$

Evaluând funcțiile și derivatele în origine se găsește că:

$$f_1(0) = f_2(0) = (1 + \beta^2)^{-3/2}, \quad (10.11)$$

$$f'_1(0) + f'_2(0) = 0, \quad (10.12)$$

$$f''_1(0) = f''_2(0) = -3(1 + \beta^2)^{-5/2} \left(1 - \frac{5\beta^2}{1 + \beta^2} \right). \quad (10.13)$$

Exercițiul 10.3: a) Verificați formulele (10.11), (10.12), (10.13).

b) Arătați că toate derivatele de ordin impar ale funcției $F(x)$ sunt zero.

Impunând și condiția ca derivata de ordinul doi a funcției $F(x)$ să fie zero² rezultă că

$$4\beta^2 = 1 \implies a = d. \quad (10.14)$$

În condiția (10.14), numită și condiția Helmholtz:

$$F(x) \approx 2f_1(0) = \frac{16}{5^{3/2}} \quad (10.15)$$

și deci, în aproximația neglijării termenilor în x^4 și de puteri mai mari (cele impare se anulează)

$$B(x) \approx 0.7155 \frac{\mu_0 I}{a} \quad (10.16)$$

nu depinde de $x = x'/a$ într-o vecinătate a punctului $x = 0$.

Exercițiul 10.4: Verificați formula (10.16).

Figura 10.2 prezintă variația mărimii B/B_0 în funcție de x'/a pentru diferite valori ale distanței d ($d = a$, $d = 1.1a$, $d = 0.9a$). Se constată că dacă este îndeplinită condiția Helmholtz (10.14) și anume **distanța dintre bobine este egală cu raza lor** atunci câmpul este practic constant în zona cuprinsă între cele două bobine.

Exercițiul 10.5: Arătați că pentru $x = \pm 0.5$ (adică în centrele spirelor) abaterea față de câmpul central este de 5.5 %.

²Pe scurt, ideea este următoarea: pentru a obține F aproximativ constantă se anulează cât mai mulți termeni din dezvoltarea Taylor.

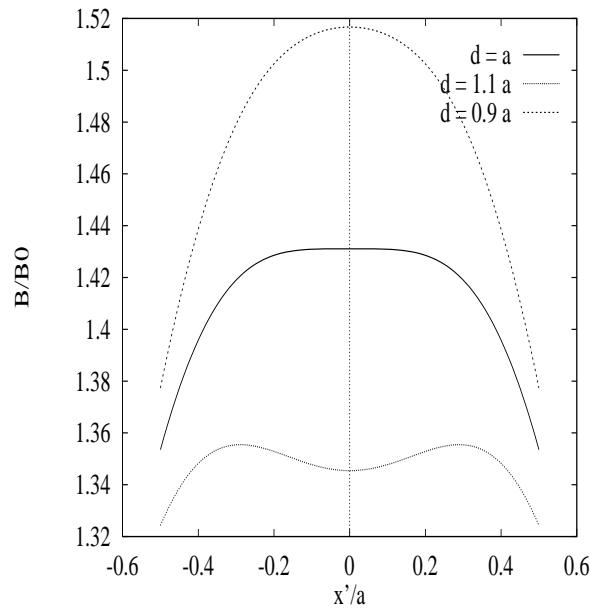


Figura 10.2: Variația B/B_0 în funcție de x'/a pentru diferite distanțe d .

Exercițiul 10.6: a) Identificați poziția spirelor în fiecare din cele trei cazuri din figura 10.2 și comparați (calitativ) omogenitatea câmpului magnetic dintre spire. Calculați aproximativ cât de mare este zona în care câmpul magnetic are o abatere de 1%.

b) Scrieți un program Scilab care să permită trasarea graficului din figura 10.2. Testați și observați omogenitatea câmpului magnetic și pentru alte valori ale raportului d/a .

10.2 Formularea problemei de optimizare

Să presupunem că bobinele Helmholtz trebuie așezate într-o regiune spațială de forma unui cub de latură $L = 30$ cm (figura 10.3) și se dorește obținerea unui câmp magnetic cât mai uniform posibil pe segmentul P_1P_2 plasat în centrul cubului, unde $|P_1P_2| = 3$ cm. Întreg spațiul se va presupune nemagnetic ($\mu = \mu_0$).

Parametrii problemei sunt în consecință raza bobinelor a și distanța dintre ele d . Curentul ce le străbate nu este necesar să intervină ca parametru. Mediul fiind liniar, câmpul magnetic depinde direct proporțional de curent, uniformitatea lui nefiind influențată de această valoare.

Exercițiul 10.7: Care sunt restricțiile de domeniu pentru parametrii a și d ?

Înainte de alegerea metodei de optimizare trebuie stabilită expresia funcției obiectiv.

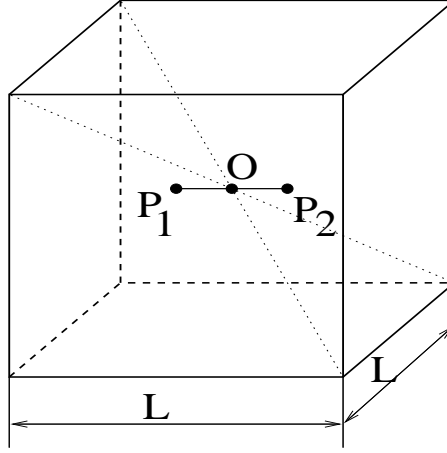


Figura 10.3: Domeniul maxim.

10.2.1 Funcție obiectiv de tip minimax

O posibilitate este de a folosi o funcție obiectiv de tipul

$$F_1(a, d) = \frac{B_{\max} - B_{\min}}{B_0}, \quad (10.17)$$

unde B_{\max} reprezintă valoarea maximă a inducției pe segmentul P_1P_2 , B_{\min} reprezintă valoarea minimă, iar B_0 este inducția în punctul O . Împărțirea la B_0 garantează faptul că rezultatul optimizării nu depinde de curentul ales (arbitrar) prin cele două spire. Problema de optimizare s-ar putea formula pe scurt astfel: “să se găsească a și d astfel încât F_1 să fie minimă”. O astfel de problemă se spune că este de tip *minimax* deoarece se dorește minimizarea abaterii maxime.

Pentru calculul mărimilor B_{\max} și B_{\min} ar putea fi folosiți de asemenea algoritmi de optimizare. Vă propunem însă ca evaluarea lor să se facă mai grosier astfel:

$$B_{\max} = \max(B(P_1), B(P_2), B(O)), \quad (10.18)$$

$$B_{\min} = \min(B(P_1), B(P_2), B(O)), \quad (10.19)$$

iar $B_0 = B(O)$.

Definirea unei funcții obiectiv de acest tip are dezavantajul că nu este continuă, deci nu poate fi derivabilă și în consecință nu se pot aplica pentru optimizare decât algoritmi de ordin zero.

Exercițiul 10.8: Rezolvați problema de optimizare folosind funcția obiectiv F_1 și cel puțin doi dintre algoritmi de ordinul zero studiați la temele 5 și 6.

Exercițiul 10.9: Rezolvați problema de optimizare folosind pentru calculul mărimilor B_{\max} și B_{\min} algoritmi de optimizare a funcțiilor unidimensionale.

10.2.2 Funcție obiectiv de tip norma Euclidiană

O altă măsură a omogenității câmpului se bazează pe abaterea pătratică (norma Euclidiană). Deoarece datorită simetriei $B(P_1) = B(P_2)$ iar B_0 este un extrem local se poate utiliza următoarea expresie a funcției obiectiv:

$$F_2(a, d) = \frac{(B(P_1) - B_0)^2}{B_0^2}. \quad (10.20)$$

În acest caz funcția obiectiv are avantajul că poate fi exprimată în mod explicit în funcție de parametrii a și d și în consecință pot fi calculate derivatele ei în raport cu parametrii de optimizat, mărimi necesare aplicării unor algoritmi de optimizare de ordin superior. În literatura de specialitate derivatele funcției obiectiv în raport cu parametrii de optimizat li se mai spun și *sensitivități*.

Exercițiul 10.10: Scrieți un program Scilab care să permită trasarea curbelor de nivel ale funcției obiectiv F_2 . Comentați “harta” obținută.

Exercițiul 10.11: Calculați sensibilitățile funcției obiectiv F_2 în raport cu parametrii a și d .

Exercițiul 10.12: Rezolvați problema de optimizare folosind funcția obiectiv F_2 și algoritmi de ordinul zero pe care i-ați folosit și la exercițiul 10.8. Comparați rezultatele obținute.

Exercițiul 10.13: Rezolvați problema de optimizare folosind funcția obiectiv F_2 și algoritmi de ordinul unu studiați la tema 7. Comparați rezultatele cu cele obținute la exercițiile 10.8 și 10.12.

Capitolul 11

Software profesional pentru rezolvarea problemelor de optimizare

În încercarea de a rezolva o problemă de optimizare cu calculatorul trebuie plecat de la observația, unanim acceptată, că nu există un program pentru calculator (cod) general, capabil să rezolve eficient orice problemă de optimizare neliniară. Datorită diversității atât a problemelor de optimizare, a algoritmilor cât și a programelor de calculator disponibile, alegerea codului potrivit pentru o problemă concretă este dificilă și cere experiență în optimizări dar și înțelegerea profundă a problemei de rezolvat.

În viața reală a cercetării științifice și ingineriei, rareori se inventează un algoritm original de optimizare și un program complet nou pentru rezolvarea unei probleme concrete, ci cel mai adesea se refolosesc coduri existente, asigurându-se în acest fel eficiența profesională în rezolvarea problemelor. În această activitate, experiența căpătată în parcurgerea capitolelor anterioare este de un real folos.

Indiferent dacă se folosesc programe de optimizare existente sau se dezvoltă coduri noi, se recomandă cu tărie ca acestea să fie verificate folosind probleme și modele de test, de preferință cât mai apropiate de problema concretă de rezolvat.

Scopul acestui capitol este de a familiariza cititorul cu principalele abordări folosite în rezolvarea cu tehnici profesionale, bazate pe reutilizarea software, a problemelor de optimizare. Spre deosebire de capitolele anterioare în care accentul era pus pe anatomia algoritmilor de optimizare, în acest capitol atenția este focalizată asupra modului în care pot fi folosite programe existente în rezolvarea unor probleme noi.

11.1 Abordări profesionale ale problemelor de optimizare

În rezolvarea profesională a problemelor de optimizare prin (re)folosirea unor coduri existente se deosebesc următoarele abordări:

- **Interactivă**, specifică unor probleme relativ simple, de mici dimensiuni, dar care trebuie rezolvate rapid (cu efort minim de dezvoltare). Se utilizează programe matematice generale ca MATLAB, Mathematica, Maple, Scilab, care au și funcții de optimizare printre comenzile de bază sau în pachete adiționale-optimale. Funcții de optimizare pot avea și alte pachete, ca de exemplu cele de tabele electronice (spreadsheet).

- **Dezvoltarea de programe în limbaje universale** folosind rutine individuale din domeniul public dedicate optimizărilor (ca de exemplu rutinele publicate în prestigioasa revistă de specialitate "Transaction on Mathematical Software"), rutine din biblioteci matematice generale sau din pachete de rutine dedicate optimizărilor, aflate att în domeniul public (ca de exemplu, depozitul Netlib, cea mai mare bibliotecă de subrutine matematice din lume menținută de Universitatea din Tennessee și disponibilă pe Internet la adresa www.netlib.org) ct și în cel comercial (dintre care cele mai cunoscute sunt bibliotecile matematice generale NAG și IMSL). De obicei aceste rutine sunt disponibile în format sursă în limbajul Fortran și, mai rar, în C, C++ sau în format obiect. În această abordare este necesar să se scrie un program principal, care să asigure interfața cu utilizatorul (citirea datelor și scrierea rezultatelor) și cel puțin o rutină care evaluează funcția obiectiv (eventual o alta pentru gradientul ei). Această abordare acordă mai multă flexibilitate și putere programatorului dect cea interactivă, motiv pentru care ea este aplicată problemelor mai complicate. În mod natural, efortul și timpul de dezvoltare sunt mai ridicate. Abordarea necesită o competență mai mare din partea utilizatorului, acesta trebuind să acorde o maximă atenție parametrilor actuali folosiți în subrutinele de bibliotecă, al căror număr, semnificație și ordine variază de la o bibliotecă la alta.

- **Descrierea problemei de optimizare în limbaje de modelare algebrică** constă în folosirea unui limbaj standard, dar specializat pentru a prezenta problema, urmnd ca interfața cu diferite pachete sau rutine de rezolvare să fie făcută de programe specializate, numite sisteme de modelare. Dintre limbajele de "modelare algebrică" cele mai răspndite sunt AMPL și GAMS. Acestea sunt în general pachete comerciale care asigură o interfață prietenoasă cu utilizatorul, dar există și sisteme de modelare plasate în domeniul public (de exemplu ASCEND, dezvoltat de Universitatea Carnegie). Scopul acestei abordări este de a elimina efortul de programare specific abordării anterioare și de a elimina riscurile aparițiilor unor erori de programare sau de interpretare. Eficiența folosirii acestor sisteme este în mod natural mai ridicată, în schimb flexibilitatea lor este mai scăzută, funcția obiectiv trebuind să admită o exprimare relativ simplă în limbaj

matematic.

• **Sevicii de optimizare disponibile pe rețea.** Rețeaua de calculatoare Internet este în afara oricărui dubiu cea mai valoroasă sursă pentru coduri de optimizare, dar și o cale de a rezolva probleme de optimizare. Mai multe site-uri Web oferă servicii de optimizare (permit evaluarea unor pachete de optimizare prin rezolvarea problemelor formulate de utilizator, fără să fie nevoie ca programul să fie instalat pe calculatorul acestuia). Dintre acestea cel mai cunoscut este serverul de optimizare NEOS, pus la dispoziție de Aragonne National Laboratory din SUA.

11.2 Surse pentru software destinat optimizării

Directorul Toms din Netlib conține subrutinele Fortran publicate în "Transaction on Mathematical Software", revistă editată de ACM (Association of Computation Machinery). Dintre acestea menționăm algoritmi dedicați rezolvării problemelor de programare pătratică (QP) care poartă numerele 559 (J.T. Betts), 587 (Hanson & Haskell) și algoritmul numărul 667 (Aluffi-Pentini), pentru minimizare globală stocastică.

Directorul Opt din Netlib conține următoarele coduri sursă Fortran77, Fortran90 sau C (de obicei convertite automat din Fortran cu utilitarul f2c):

- **connax.f** - minimizarea funcțiilor neliniare cu restricții (E.H. Kaufman);
- **donlp2** - optimizare neliniară (cu restricții neliniare), cu derivate și matrice pline prin metoda de tip SQP (P.Spellucci);
- **dqed.f** - cele mai mici pătrate neliniare, cu restricții liniare (R. Hanson);
- **hooke.c** - optimizare fără restricții, fără derivate cu metoda Hooke-Jeeves (M.G. Johnson);
- **lbfgs** - optimizare neliniară fără restricții sau cu restricții de tip frontieră, prin metoda BFGS (J.Nocedal);
- **lsnno** - optimizare neliniară cu restricție liniară de tip "rețea" (P. Toint);
- **praxis** - optimizare fără restricții și fără derivate, cu metoda "axelor principale" (R.Brent);
- **simannf** - optimizare cu restricții simple prin metoda "Simulated Annealing" (B. Goffe);
- **subplex** - optimizare fără restricție prin metoda căutării simplex în subspațiu (T. Rowan)

- **tn** - optimizare fără restricții sau cu restricții de tip frontieră prin metoda Newton (S. Nash);
- **varpov** - cele mai mici pătrate neliniare separabile (Bolstad);
- **ve08** - optimizare fără restricții a funcțiilor separabile (P. Toint).

Alte pachete de programare din domeniul public sau "shareware" disponibile în format sursă sunt:

- **TRON** - probleme de optimizare cu restricții de tip frontieră prin metoda Newton cu "trust-region" (Lin și More);
- **L-BFGS-B** - probleme de optimizare cu restricțiile de tip frontieră (Zhu și Nocedal);
- **conmin** - pachet pentru probleme de optimizare neliniară cu restricții (Murry Dow);
- **WNLIB** - rutine pentru probleme de optimizare neliniară cu și fără restricție prin metode de gradient, dar conțin și rutine de tip "Simulated Annealing" (Will Naylor);
- **SolvOpt** - optimizări locale în probleme neliniare cu funcții nederivabile (A. Kuntserich, F. Kappel);
- **FFSQP/CFSQP** - cod Fortran/C pentru probleme neliniare cu restricții prin programări pătratice succesive (A. Tits);
- **MINIPACK** - cod pentru rezolvarea problemei celor mai mici pătrate neliniare (J. More);
- **BARON** - module pentru optimizare globală neliniară în probleme cu variabile continui și/sau discrete (N. Sahhinidis);
- **cGOP** - pachet de funcții C pentru optimizare globală cu tehnici de tip "branch-and-bound" (Computer Aided System Lab at Princetown University);
- **GA Playground** - un applet Java ce permite rezolvarea problemelor de optimizare cu Algoritmi Genetici prin browsere Web;
- **Adaptive Simulated Annelig** (L. Ingber) și **Ensemble Based Simulated Annealing** - pachete de optimizare globală prin metoda "simann" (R. Frost);
- **GENOCOP** - program de algoritmi genetici (Michalewicz).

Produsul	Metoda	Vănzătorul	Adresa web
GAUSS,CML	SQP ³	APTECH	http://www.aptech.com
CONOPT ¹	GRG ⁴	ARKI	
Multisimplex	Nelder-Mead	Multisimplex	http://www.multisimplex.com
NAG Num Lib.	diverse	NAG	http://www.nag.com
GRG2, SLP, SQP	diverse	Optimal Methods	http://www.optimalmethods.com
LGO	diverse	Pinter Consulting	http://is.dal.ca/jdpinter
LANCELOT	Trust-region	Rutherford Lab.(edu. free)	http://www.dci.clrc.ac.uk/Activity/LANCELOT
SOPT	SQP	SAITECH	http://www.saitech-inc.com
MINOS, SNOPT	SQP	Stanford Bus Soft	http://www.sbsi-sol-optimize.com
FSQP	SQP	Tits (edu. free)	http://www.isr.umd.edu/Labs/CACSE/FSQP
IMSL ²	diverse	Visual Numerics	http://www.vni.com/products/imsi
CONMIN, DOC, DOT	diverse	Vanderploats	ftp://anusf.anu.edu.au/mld900/constr_minimum

Tabelul 11.1: Coduri comerciale (cu taxă pentru licența de utilizare) de optimizare.

Detalii asupra acestor pachete sunt disponibile la <http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html> și la adresele din tabelul 11.1.

Următoarele site-uri web oferă servicii de optimizare on-line prin rețea:

- **AMPL** (<http://www.ampl.com/ampl>) - permite rezolvarea problemelor cu pnă la 350 variabile, descrise în limbaj AMPL, prin 8 solveere liniare sau neliniare (inclusiv cu variabile discrete);
- **NEOS** (<http://www-neos.mcs.anl.gov>) - oferă servicii de acces la peste o duzină de solveere de optimizare liniară, neliniară, stocastică, cu sau fără restricții. Problemele pot fi descrise în C, Fortran sau în limbaje de modelare ca AMPL sau GAMS
- **UniCalc** (<http://www.rriai.org.ru/UniCalc>) - O mare varietate de probleme de optimizare trimise printr-un formular disponibil pe web. Abordarea se face în limbajul de modelare UniCalc ce permite combinarea unor concepte novatoare, cum sunt: matematica intervalurilor, algebra simbolică, strategii originale de căutare și o interfață prietenoasă cu utilizatorul.

Metodele de optimizare implementate în rutinele și pachetele software profesionale sunt de o mare diversitate. Ele se împart în două mari categorii, prima, bazată pe metode

¹Admite descrieri în limbaje de modelare AMPL, GAMS, AIMMS și LINGO.

²Admite interfață opțională pentru AMPL.

³SQP = Sequential Quadratic Programming.

⁴GRG = Generalised Reduced Gradient.

deterministe, este dedicată optimizărilor locale, iar a doua, bazată pe metode stocastice, este dedicată optimizărilor globale. Principiile care stau la baza acestor metode sunt rezumate în anexele D (metode deterministe) și E (metode stocastice).

11.3 Rezolvarea interactivă a problemelor simple

În mediul Scilab există o funcție dedicată optimizării neliniare, bazată pe metoda quasi-Newton. Această funcție, numită **optim**, are următoarea secvență simplă de apel (pentru optimizări fără restricție):

$$[f, xopt] = \text{optim}(\text{costf}, x0),$$

în care: **costf** este numele funcției obiectiv ce trebuie minimizată, **xopt** este vectorul soluție a problemei de optimizare, **f** este valoarea optimă a funcției obiectiv $f = \text{costf}(xopt)$ iar **x0** este inițializarea soluției (vector de aceeași dimensiune cu **xopt**).

Funcția obiectiv are următorul prototip:

$$[f, g, ind] = \text{costf}(x, ind),$$

în care **f** este valoarea funcției obiectiv, **g** este vectorul gradient al funcției obiectiv în punctul **x** (un vector care reprezintă variabila independentă), iar **ind** este un indicator întreg de eroare (la ieșire) sau de comandă (la intrare), cu următoarea semnificație:

ind < 0 (la ieșire)	- <i>f</i> nu poate fi evaluat în <i>x</i> ;
ind = 0	- se întrerupe optimizarea;
ind = 1	- nu se calculează nimic;
ind = 2	- se calculează numai <i>f</i> ;
ind = 3	- se calculează numai <i>g</i> ;
ind = 4	- se calculează și <i>f</i> și <i>g</i> .

Exemplu de utilizare:

```
x0=[0;0;0]
xa=[0;1;2]
deff(f,g,ind)=cost(x,ind)=norm(x-xa)^2, g=2*(x-xa)
[f,xopt]=optim(cost, x0)
```

Exercițiul 11.1: *Ce funcție se minimizează în exemplul de mai sus? Verificați exemplul, lucrând în consola Scilab.*

Trebuie remarcat că Scilab admite ca funcția de cost să fie descrisă și în limbajul Fortran. Detalii asupra sintaxei în acest caz sunt prezentate în manualul de utilizare Scilab [8] și în sistemul de asistență on-line Scilab/help.

Exercițiul 11.2: *Scrieți un program Scilab care să minimizeze o funcție pătratică $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c$, unde \mathbf{A} este o matrice pătrată de dimensiune $n \times n$, \mathbf{b} este un vector coloană de dimensiune n iar c este o constantă reală. Aplicație numerică pentru $c = 10$, $\mathbf{b} = [1; -1; 0]$, \mathbf{A} matricea unitate.*

Scilab admite și un mod mai complicat de apel al comenzii `optim`, mod care se aplică în cazul optimizărilor cu restricții, și care are sintaxa:

```
[f, [xopt, [gradopt, [work]]]] = optim(costf, [contr], x0, ['algo'], [work], [stop]),
```

în care intervin în plus următoarele variabile: variabila `gradopt` este gradientul funcției `costf` în punctul `xopt`; `work` este tablou de lucru utilizat la restartul algoritmilor quasi-Newton (este inițializat automat de `optim`); `constr` descrie restricțiile cu sintaxa: '`b`', `binf`, `bsup` unde `binf` și `bsup` sunt vectori de dimensiunea vectorului soluție, ce conțin marginile inferioară și superioară ale componentelor variabilei independente \mathbf{x} , '`algo`' indică algoritmul utilizat și poate fi '`qn`' sau '`gc`' sau '`nd`', fiecare valoare corespunzând respectiv metodelor quasi-Newton, gradienti conjugați și metodă de calcul fără derivată (pentru funcții nediferențiabile). În acest din urmă caz nu se acceptă restricții. Parametrul `stop` este o secvență de parametri opționali care controlează criteriul de oprire al algoritmului, cu sintaxa: '`ar`', `nap`, [`iter` [, `epsg` [, `epsf` [, `epsx`]]]] în care '`ar`' este un cuvânt cheie rezervat pentru selecția criteriului de oprire, `nap` este numărul maxim de apeluri pentru funcție, `iter` este numărul maxim de iterații admise, `epsg` este toleranța normei gradientului, `epsf` este toleranța descreșterii valorii funcției obiectiv, `epsx` este toleranța variației vectorului \mathbf{x} (vector de dimensiunea lui `x0`).

Exemplu de utilizare:

```
[f, xopt, gopt] = optim(cost, x0, c
[f, xopt, gopt] = optim(cost, [-1, 0, 2], [0, 1, 4], x0)
[f, xopt, gopt] = optim(cost, [-1, 0, 2], [0, 1, 4], x0, 'gc', 'ar', 3)
```

Exercițiul 11.3: *Comentați cele trei instrucțiuni de mai sus. Executați în consola Scilab cele trei instrucțiuni din exemplul de mai sus, precedate de instrucțiunile din exercițiul 11.1.*

Exercițiul 11.4: Scrieți un program Scilab care să minimizeze funcția lui Rosenbrock cu ajutorul procedurilor de optimizare furnizate de program. Comparați rezultatele cu cele obținute pe parcursul temelor anterioare, în care ați implementat proceduri proprii de minimizare.

Pentru optimizări liniare și pătratice, Scilab are comenzi specializate `linpro`, și `quapro`.

11.4 Utilizarea rutinelor din biblioteci matematice

În continuare va fi prezentat și analizat modul în care pot fi folosite rutinele de optimizare din bibliotecile matematice. Pentru început va fi testată funcția `hooke.c` din biblioteca `netlib/org`. Aceasta este scrisă în limbajul C și permite determinarea unui minim local al funcției $f : \mathbb{R}^n \rightarrow \mathbb{R}$, prin metoda Hooke-Jeeves (“pattern search”). Aceasta este o metodă de ordinul zero, în consecință ea poate fi aplicată și funcțiilor nederivabile și discontinue.

Exercițiul 11.5: a) Folosind un browser de Internet, ca de exemplu Netscape, inspectați directorul de optimizări din biblioteca `netlib` la adresa <http://www.netlib.org/opt> și vizualizați fișierul **hooke.c** (citiți comentariul introductiv și identificați funcțiile componente ale codului sursă: **f**, **best-nearby**, **hooke** și **main**).

b) Salvați fișierul **hooke.c** în directorul de lucru. Completați fișierul și apoi executați programul folosind sub controlul sistemului de operare comenzile: **make hooke** și apoi **hooke**.

c) Modificați funcția obiectiv (folosind editorul Dvs. preferat) și executați din nou programul. Care este soluția obținută în urma minimizării funcției lui Rosenbrock? Cte evaluări ale funcției obiectiv au fost necesare pentru obținerea acestui rezultat? Analizați ce soluție se obține, în funcție de inițializarea folosită. Ce efect au ceilalți parametri (**rho**, **epsilon** și **itermax**)?

11.5 Utilizarea serviciului de optimizare prin Internet

Există și servicii de optimizare prin Internet. Unul dintre acestea este NEOS (Network Enabled Optimization System). Acest serviciu este organizat de Centrul de Tehnologie Lab și Northwestern University din SUA și este disponibil la adresa: <http://www-fp.mcs.anl.gov/otc/index.html>.

Pentru utilizarea acestui serviciu va trebui să vă alegeți ca solver o metodă de optimizare. Metodele de optimizare sunt grupate în următoarele categorii:

- optimizare discretă;
- optimizare neliniară cu restricții;
- optimizare cu restricții de tip frontieră;
- optimizare fără restricții;
- programare liniară;
- programare liniară stocastică;
- probleme complementare;
- optimizări de rețele liniare;
- programare semidefinită.

Exercițiul 11.6: Folosiți serviciul de optimizare NEOS pentru optimizarea oricăreia din problemele descrise în capitolele anterioare. Comparați rezultatele și performanțele programului de optimizare cu cele obținute de dvs. la capitolele respective.

Anexa A

Tipuri de probleme de optimizare

A.1 Optimizări scalare

În cele ce urmează vom nota cu $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ funcția obiectiv (reală), cu $\mathbf{x} \in \mathbb{R}^n$ vectorul variabilelor, restricțiile sunt reprezentate de funcția $c : \mathbb{R}^n \rightarrow \mathbb{R}^p$ unde unele componente ale lui c sunt restricții de tip inegalitate, de forma $c_i(\mathbf{x}) \leq 0$ pentru i într-o mulțime de indici notată \mathcal{I} , alte componente ale lui c sunt restricții de tip egalitate de tip $c_i(\mathbf{x}) = 0$ pentru indici i dintr-o mulțime de indici notată \mathcal{E} . Pot exista restricții bilaterale de tipul $l_i \leq c_i(\mathbf{x}) \leq u_i$ unde l_i și u_i sunt marginile inferioară și superioară ale restricției, cazul $c_i(\mathbf{x}) = x_i$ corespunzând restricțiilor de domeniu.

Iată tipurile de probleme de optimizare [47]:

1. Probleme de programare neliniară cu restricții¹

În aceste probleme f_0 și c sunt funcții neliniare de \mathbf{x} . Două formulări tipice echivalente sunt

$$\min \{f_0(\mathbf{x}) | c_i(\mathbf{x}) \leq 0, i \in \mathcal{I}, c_i(\mathbf{x}) = 0, i \in \mathcal{E}\},$$

și

$$\min \{f_0(\mathbf{x}) | c(\mathbf{x}) = \mathbf{0}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}.$$

2. Probleme de programare liniară²

În aceste probleme funcția obiectiv și restricțiile sunt liniare. Formularea standard a acestor probleme este:

$$\min \{ \mathbf{c}^T \mathbf{x} | \mathbf{A} \mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \},$$

¹ "Nonlinear programming"

² "Linear programming"

unde $\mathbf{c} \in \mathbb{R}^n$ este un vector de cost și $\mathbf{A} \in \mathbb{R}^{m \times n}$ este o matrice de restricții. De multe ori se folosește formularea mai convenabilă

$$\min \{ \mathbf{c}^T \mathbf{x} \mid l_i \leq \mathbf{a}_i^T \mathbf{x} \leq u_i, i \in \mathcal{I}, l_i \leq x_i \leq u_i, i \in \mathcal{B} \}.$$

În această formulare restricțiile de tip egalitate sunt descrise prin relații de încadrare în care $l_i = u_i$ pentru $i \in \mathcal{I}$.

3. Programare pătratică³

Aceste probleme au restricții liniare și funcții obiectiv pătratice, formularea problemei făcându-se astfel

$$\min \left\{ \mathbf{c}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} \mid \mathbf{a}_i^T \mathbf{x} \leq b_i, i \in \mathcal{I}, \mathbf{a}_i^T \mathbf{x} = b_i, i \in \mathcal{E} \right\},$$

unde $\mathbf{Q} \in \mathbb{R}^{n \times n}$ este o matrice simetrică. Aceste probleme sunt convexe dacă \mathbf{Q} este pozitiv semidefinită și neconvexe în caz contrar.

4. Probleme de cele mai mici pătrate cu restricții liniare⁴

Unele probleme convexe de programare pătratică pot fi formulate mai natural ca o problemă de cele mai mici pătrate

$$\min \left\{ \frac{1}{2} \|\mathbf{C}\mathbf{x} - \mathbf{d}\|_2^2 \mid \mathbf{a}_i^T \mathbf{x} \leq b_i, i \in \mathcal{I}, \mathbf{a}_i^T \mathbf{x} = b_i, i \in \mathcal{E} \right\},$$

unde matricea coeficienților \mathbf{C} nu este neapărat pătrată.

5. Probleme care au doar restricții de domeniu⁵

În aceste probleme singurele restricții sunt cele care mărginesc componentele vectorului \mathbf{x}

$$\min \{ f_0(\mathbf{x}) \mid \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \}.$$

6. Probleme fără restricții⁶

sunt cele care nu au nici un fel de restricții (nici măcar de domeniu).

7. Probleme neliniare de cele mai mici pătrate⁷

În aceste probleme funcția obiectiv are forma specială

$$f_0(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^m f_i(\mathbf{x})^2,$$

³ "Quadratic Programming"

⁴ "Constrained linear least square problems"

⁵ "Bound Constrained Problems"

⁶ "Unconstrained problems"

⁷ "Nonlinear least square problems"

unde fiecare componentă f_i este numită reziduu. Grupând reziduurile într-o funcție vectorială $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ putem scrie f_0 ca fiind

$$f_0(\mathbf{x}) = \frac{1}{2} \|f(\mathbf{x})\|_2^2.$$

8. Sisteme de ecuații neliniare

Rezolvarea unui sistem de ecuații neliniare definit cu ajutorul funcției $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ este un vector \mathbf{x} astfel încât $f(\mathbf{x}) = \mathbf{0}$. Unii algoritmi sunt stânși legați de algoritmi de optimizare fără restricții și de cele mai mici pătrate neliniare, ei rezolvând

$$\min \{ \|f(\mathbf{x})\| \mid \mathbf{x} \in \mathbb{R}^n \},$$

unde $\|\cdot\|$ este de obicei norma L_2 din \mathbb{R}^n .

9. Probleme de optimizare a rețelelor ⁸

În aceste probleme f_0 și c_i au o structură specială care provine de la un graf constând în arce și noduri. Astfel de probleme apar în aplicații care implică distribuția produselor, transportul, comunicațiile. Restricțiile sunt de obicei liniare și fiecare restricție c_i implică de obicei numai una sau două componente ale lui \mathbf{x} . Funcția f_0 poate fi fie liniară fie neliniară, dar este de obicei "separabilă", adică se poate scrie sub forma

$$f_0(\mathbf{x}) = \sum_{i=1}^n f_i(x_i),$$

unde fiecare funcție scalară f_i depinde numai de argumentul x_i .

10. Programare întreagă ⁹

Acest termen definește problemele de optimizare în care componentele lui \mathbf{x} sunt întregi. Problemele de tip 1÷9 au presupus valori reale ale componentelor. Termenul *programare mixtă* ¹⁰ definește problemele în care o parte a componentelor lui \mathbf{x} sunt reale iar cealaltă parte sunt întregi. Astfel de probleme sunt mai dificil de rezolvat, de aceea până acum au apărut metode care rezolvă doar problemele de programare liniară și unele cazuri de programare pătratică. Aceste metode se bazează pe tehnica "branch and bound".

Majoritatea problemelor de optimizare nu se pot încadra strict într-una din categoriile de mai sus. De exemplu o problemă de programare liniară este în același timp o problemă de programare pătratică care este în același timp o problemă de programare neliniară. O problemă trebuie plasată în categoria cea mai restrictivă pentru că astfel se pot folosi metode și implementa algoritmi care să țină cont de toate particularitățile problemei.

⁸"Network optimization problems"

⁹"Integer programming"

¹⁰"Mixed-integer programming"

A.2 Optimizări vectoriale

Una din dificultățile întâmpinate în optimizarea dispozitivelor electromagnetice constă în cerința de a satisface mai multe obiective. Problemele care urmăresc satisfacerea simultană a mai multor obiective se numesc probleme de optimizări vectoriale.

A.2.1 Optimalitate Pareto

Caracteristic pentru problemele de optimizare vectorială este apariția unui conflict de obiective, în care soluțiile care ar minimiza fiecare obiectiv în parte sunt diferite și nu există soluție acolo unde toate obiectivele își ating minimumul. O problemă de minimizare vectorială se formulează astfel:

Să se minimizeze $\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_K(\mathbf{x}))$ unde $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^K$, este supusă la restricțiile $g_i, h_j : \mathbb{R}^n \rightarrow \mathbb{R}$, $g_i(\mathbf{x}) \leq 0, (i = 1, 2, \dots, m)$ (restricții de tip inegalități), $h_j(\mathbf{x}) = 0, (j = 1, 2, \dots, p)$ (restricții de tip egalități) și $x_{lu} \leq x_l \leq x_{lu}, (l = 1, 2, \dots, n)$ (restricții de domeniu).

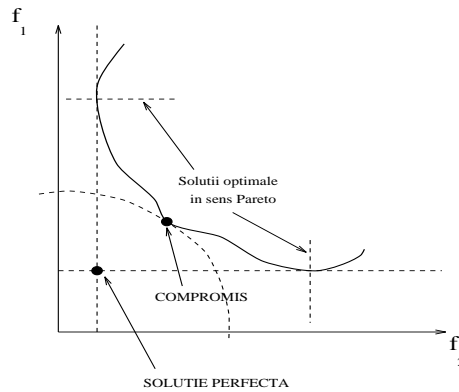


Figura A.1: Interpretarea geometrică a soluțiilor optime în sens Pareto

În definirea soluției optime se aplică criteriul de optimalitate introdus inițial de Pareto încă din 1896 pentru problemele din economie [55]. O soluție optimală \mathbf{x}^* în sens Pareto se caută acolo unde nu există o soluție \mathbf{x} în domeniul de căutare $M = \{\mathbf{x} \in \mathbb{R}^n | g_i(\mathbf{x}) \leq 0; h_j(\mathbf{x}) = 0; x_{lu} \leq x_l \leq x_{lu} (\forall i = 1, \dots, m; j = 1, \dots, p; l = 1, \dots, n)\}$ pentru care $f_k(\mathbf{x}) \leq f_k(\mathbf{x}^*), (\forall k \in [1, K], f_k(\mathbf{x}) < f_k(\mathbf{x}^*), \text{ pentru cel puțin un } k \in [1, K]$. O problemă de optimizare în care îmbunătățirea unui obiectiv cauzează degradarea a cel puțin unui alt obiectiv nu are soluție decât în sens optimal Pareto. Figura A.1 arată interpretarea geometrică a soluției optime în sens Pareto pentru cazul a două obiective care sunt în conflict.

Graficul din figură reprezintă dependența dintre f_1 și f_2 . Pe această curbă există soluțiile posibile ale problemei, fiind marcată regiunea soluțiilor optimale în sens Pareto. Se observă că nu există un punct pentru care ambele obiective să își atingă minimul. Acel punct este numit "soluție perfectă". O soluție de compromis ar putea fi aceea pentru care distanța dintre soluția perfectă dar nefezabilă și mulțimea soluțiilor optimale în sens Pareto este minimă.

Abordarea unei probleme de optimizări vectoriale are trei aspecte importante: stabilirea funcției obiectiv ¹¹, metodele de a trata restricțiile neliniare și alegerea algoritmului de optimizare care minimizează funcția obiectiv.

A.2.2 Stabilirea funcției obiectiv în cazul optimizărilor vectoriale

Aplicarea unui algoritm de optimizare are nevoie de o metodă de luare a deciziilor care garantează o soluție din mulțimea de soluții optimale în sens Pareto. Iată câteva metode care se aplică în optimizarea dispozitivelor electromagnetice [54, 55].

• Ponderarea obiectivelor

O primă posibilitate este aceea de a se folosi o funcție obiectiv care este suma ponderată a tuturor obiectivelor de minimizat. Noua funcție obiectiv este $u(\mathbf{F}(\mathbf{x})) = \sum_{k=1}^K t_k f_k(\mathbf{x})$ unde $x \in M$. Pentru probleme de optimizare în care toate funcțiile de cost f_k sunt convexe se poate arăta că problema minimizării funcției u are o soluție optimală în sens Pareto unică [55]. Problema care apare constă în găsirea unor ponderi potrivite, cunoscând faptul că obiectivele au valori numerice diferite și sensibilități diferite. Ponderarea obiectivelor este de aceea un proces iterativ în care trebuie făcute mai multe optimizări, cu ponderi recalculat.

• Ponderarea distanțelor

O a doua posibilitate este de folosi metoda "funcției distanță". Presupunem că f_k^* sunt cerințele de atins (minimele funcțiilor de cost f_k). Funcția obiectiv care se folosește este o distanță (de cele mai multe ori în sensul celor mai mici pătrate). Și aici apare problema stabilirii ponderilor, funcția de minimizat fiind $\|\mathbf{z}(\mathbf{x})\|^2 = \sum_{k=1}^K t_k (f_k^*(\mathbf{x}) - f_k(\mathbf{x}))^2$. Pentru funcții convexe și dacă f_k^* sunt minimele fiecărei funcții obiectiv se poate arăta că $\|\mathbf{z}\|$ are un optim unic în sens Pareto. Dezavantajul folosirii unei astfel de norme euclidiene este sensibilitatea scăzută la reziduuri subunitare. De aceea trebuie folosiți factori de pondere suficient de mari.

• Reformularea problemei cu restricții

Problema factorilor de pondere poate fi depășită reformulând problema astfel: numai unul

¹¹Procesului de stabilire a funcției obiectiv i se mai spune și "alegerea criteriului de decizie" folosit de algoritmul de optimizare propriu-zis.

din obiective este minimizat, celelalte fiind considerate restricții suplimentare. Problema astfel reformulată minimizează doar $f_i(\mathbf{x})$ (cu i fixat) cu restricțiile suplimentare $f_k(\mathbf{x}) - r_k \leq 0, (\forall) k = 1, K, k \neq i$. Valoarea r_k reprezintă minimumul cerut pentru funcția de cost k . O astfel de formulare are avantajul că ei i se poate aplica o tehnică de tip Lagrange.

Anexa B

Funcții de test pentru algoritmi de optimizare

Pentru a testa algoritmi de optimizare se folosesc funcții de test cu expresii analitice, care au punctele de extrem cunoscute. Această anexă cuprinde cele mai cunoscute funcții de test [42], [65].

B.1 Probleme care au doar restricții de domeniu

1. De Jong (I):

$$\sum_{i=1}^3 x_i^2,$$

unde $-5.12 \leq x_i \leq 5.12$, $i = 1, 3$. Funcția are un minim global egal cu 0 în punctul $(0, 0, 0)$.

2. De Jong (II):

$$\sum_{i=1}^5 \text{integer}(x_i),$$

unde $-5.12 \leq x_i \leq 5.12$, $i = 1, 5$. Funcția are un minim global egal cu -30 pentru toți $-5.12 \leq x_i < -5.0$, $i = 1, 5$.

3. De Jong (III):

$$\sum_{i=1}^{30} ix_i^4 + \text{Gauss}(0, 1),$$

unde $-1.28 \leq x_i \leq 1.28$, $i = 1, 30$. Funcția (fără zgomot Gaussian) are un minim global egal cu 0 în punctul $(0, \dots, 0)$.

4. Schaffer (I):

$$0.5 + \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0.5}{[1.0 + 0.001(x_1^2 + x_2^2)]^2},$$

unde $-100 \leq x_i \leq 100$, $i = 1, 2$. Funcția are un minim global egal cu 0 în punctul $(0, 0)$.

5. Schaffer (II):

$$(x_1^2 + x_2^2)^{0.25} [\sin^2(50(x_1^2 + x_2^2)^{0.1} + 1.0)],$$

unde $-100 \leq x_i \leq 100$, $i = 1, 2$. Funcția are un minim global egal cu 0 în punctul $(0, 0)$.

6. Goldstein-Price:

$$[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\ \cdot [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)],$$

unde $-2 \leq x_i \leq 2$, $i = 1, 2$. Funcția are un minim global egal cu 3 în punctul $(0, -1)$.

7. Branin:

$$a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e,$$

unde $-5 \leq x_1 \leq 10$, $0 \leq x_2 \leq 15$, și $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $d = 6$, $e = 10$, $f = 1/(8\pi)$. Funcția are un minim global egal cu 0.397887 în trei puncte diferite: $(-\pi, 12.275)$, $(\pi, 2.275)$, și $(9.42478, 2.475)$.

8. Shubert:

$$\sum_{i=1}^5 i \cos[(i+1)x_1 + i] \cdot \sum_{i=1}^5 i \cos[(i+1)x_2 + i],$$

unde $-10 \leq x_j \leq 10$, $j = 1, 2$. Funcția are 760 de minime locale, dintre care 18 sunt minime globale de valoare -186.73.

9. Easom:

$$-\cos(x_1) \cos(x_2) e^{-(x_1 - \pi)^2 - (x_2 - \pi)^2},$$

unde $-100 \leq x_i \leq 100$ pentru $i = 1, 2$. Funcția are un minim global egal cu -1 în punctul (π, π) .

10. Bohachevsky (I):

$$x_1^2 + 2x_2^2 - 0.3 \cos(3\pi x_1) - 0.4 \cos(4\pi x_2) + 0.7,$$

unde $-50 \leq x_i \leq 50$, $u = 1, 2$. Funcția are un minim global egal cu 0 în punctul $(0, 0)$.

11. Bohachevsky (II):

$$x_1^2 + 2x_2^2 - 0.3 \cos(3\pi x_1) \cos(4\pi x_2) + 0.3,$$

unde $-50 \leq x_i \leq 50$, $i = 1, 2$. Funcția are un minim global egal cu 0 în punctul $(0, 0)$.

12. Bohachevsky (III):

$$x_1^2 + 2x_2^2 - 0.3 \cos(3\pi x_1) + \cos(4\pi x_2) + 0.3,$$

unde $-50 \leq x_i \leq 50$, $i = 1, 2$. Funcția are un minim global egal cu 0 în punctul $(0, 0)$.

13. Colville:

$$100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + \\ + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1),$$

unde $-10 \leq x_i \leq 10$, $i = 1, 4$. Funcția are un minim global egal cu 0 în punctul $(1, 1, 1, 1)$.

14. Cămila cu șase cocoșe;

$$\left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

unde $-3 \leq x_1 \leq 3$ și $-2 \leq x_2 \leq 2$. Funcția are un minim global egal cu -1.03163 în două puncte diferite: $(-0.0898, -0.7126)$ și $(0.0898, -0.7126)$. Harta funcției pe domeniul $[0, 2] \times [-1.2, 1]$ este prezentată în figura B.1.

15. Schwefel (I)

$$\sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|,$$

unde $-10 \leq x_i \leq 10$, $i = 1, 30$. Funcția are un minim global egal cu 0 în punctul $(0, \dots, 0)$.

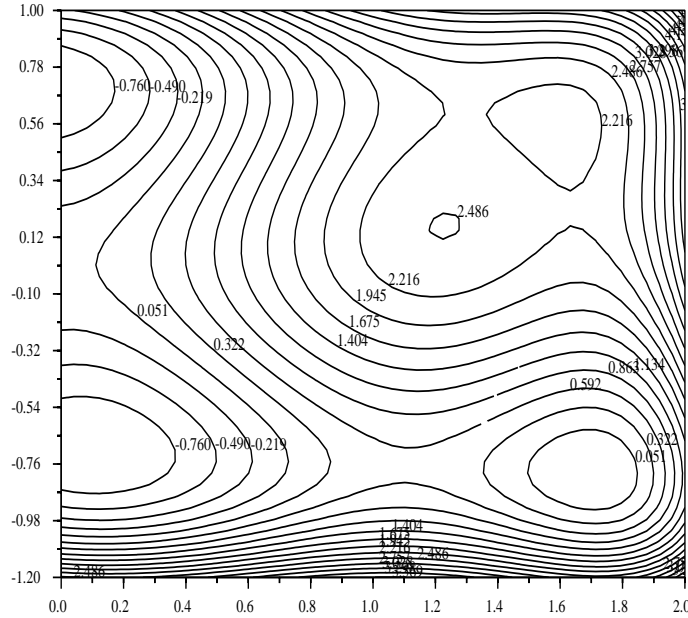


Figura B.1: Harta funcției “cămila”.

16. Schwefel (II)

$$\sum_{i=1}^{30} \left(\sum_{j=1}^i x_j \right)^2,$$

unde $-100 \leq x_i \leq 100$, $i = 1, 30$. Funcția are un minim global egal cu 0 în punctul $(0, \dots, 0)$.

17. Schwefel (III)

$$-\sum_{i=1}^{30} (x_i \sin(\sqrt{|x_i|})),$$

unde $-500 \leq x_i \leq 500$, $i = 1, 30$. Funcția are un minim global egal cu -12569.5 în punctul $(420.9687, \dots, 420.9687)$. Harta funcției în cazul bidimensional, pe domeniul $[0, 500] \times [0, 500]$ este prezentată în figura B.2.

18. Rosenbrock (“banana”):

$$100(x_1^2 - x_2)^2 + (1 - x_1)^2,$$

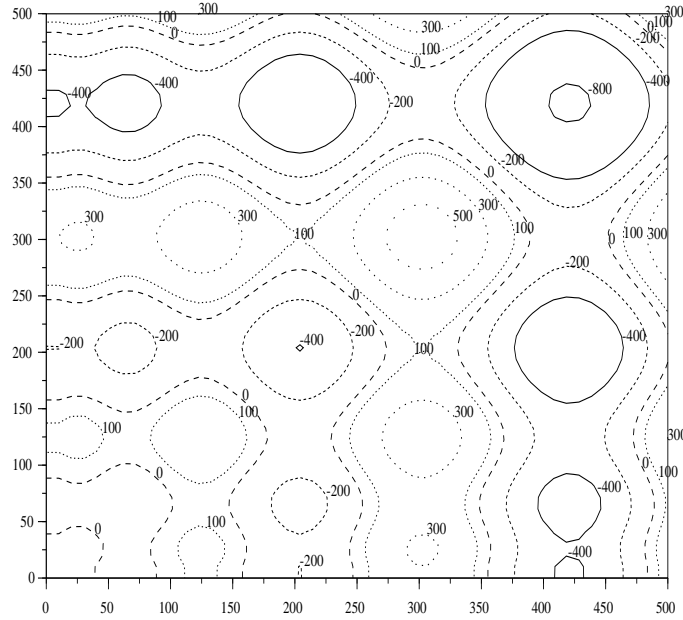


Figura B.2: Harta funcției Schwefel (III).

unde $-2.048 \leq x_i \leq 2.048$, $i = 1, 2$. Funcția are un minim global egal cu 0 în punctul $(1, 1)$. Harta funcției este prezentată în figura B.3.

19. Rosenbrock generalizată:

$$\sum_{i=1}^{29} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$

unde $-30 \leq x_i \leq 30$, $i = 1, 30$. Funcția are un minim global egal cu 0 în punctul $(1, \dots, 1)$.

20. Rastrigin generalizată:

$$\sum_{i=1}^{30} [x_i^2 - 10 \cos(2\pi x_i) + 10],$$

unde $-5.12 \leq x_i \leq 5.12$, $i = 1, 30$. Funcția are un minim global egal cu 0 în punctul $(0, \dots, 0)$.

21. Griewank generalizată:

$$\frac{1}{4000} \sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1,$$

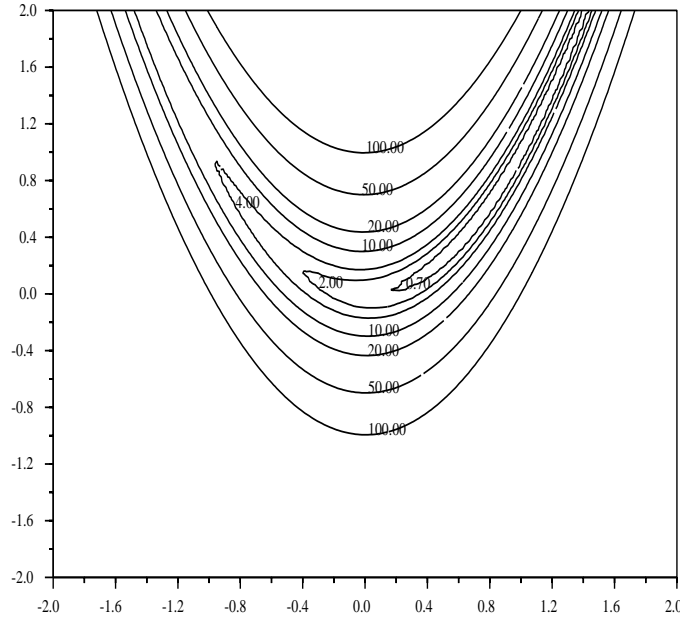


Figura B.3: Harta funcției Rosenbrock (banana”).

unde $-600 \leq x_i \leq 600$, $i = 1, 30$. Funcția are un minim global egal cu 0 în punctul $(0, \dots, 0)$.

22. Vizuina vulpii:

$$\left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + (x_1 - a_{1j})^6 + (x_2 - a_{2j})^6} \right)^{-1},$$

unde $-65.536 \leq x_i \leq 65.536$, $i = 1, 2$ și

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}.$$

Funcția are un minim global egal cu 0.998 în punctul $(-32, -32)$ și alte minime locale în restul de 24 de puncte (a_{1j}, a_{2j}) . Graficul funcției este prezentat în figura B.4.

23. Funcții deceptive:

Să ne imaginăm că dorim să maximizăm o funcție reală de două variabile $F(x, y)$, construită astfel. Fie k un număr de maxime globale plasate arbitrar. Pentru a

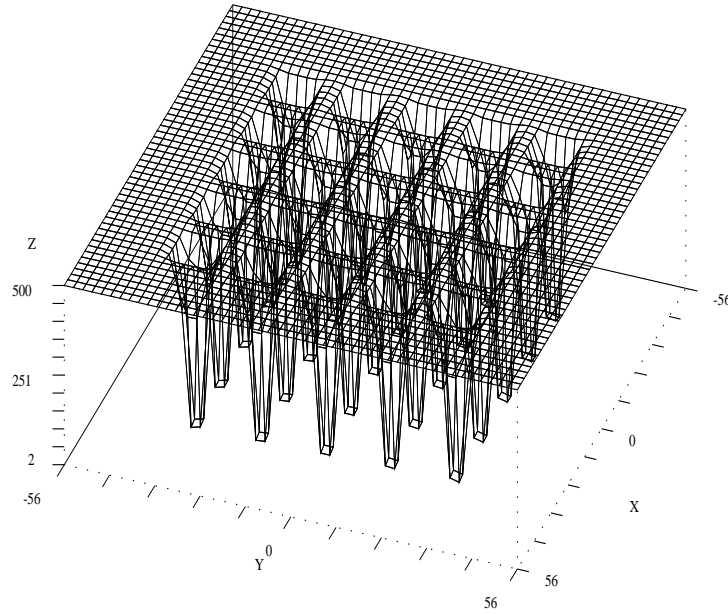


Figura B.4: Graficul funcției “viziunea vulpii”.

ilustra acest exemplu vom considera $k = 5$ și optimele globale vor fi plasate în punctele mulțimii:

$$G = \{(7, 59), (5, 21), (30, 7), (62, 3), (62, 51)\},$$

funcția F fiind definită pe domeniul $[0, 63] \times [0, 63]$. Să considerăm acum funcția $f_0(x, y)$ definită pe același domeniu, dată de relația:

$$f_0(x, y) = \min_{\forall (x_g, y_g) \in G} \sqrt{(x_g - x)^2 + (y_g - y)^2}. \quad (\text{B.1})$$

Funcția f_0 reprezintă distanța de la punctul (x, y) la mulțimea G . Această funcție are un maxim egal cu 32.0156 în punctul $(32, 39)$. Funcția F este definită de relația:

$$F(x, y) = \begin{cases} 34 & \text{dacă } (x, y) \in G \\ f_0(x, y) & \text{dacă } (x, y) \notin G \end{cases}$$

Graficul acestei funcții este prezentat în figura B.5. Peisajul ei este multimodal, cu puncte de atracție false pentru algoritmi determiniști de ordin superior. Punctul de atracție cel mai puternic este cel care corespunde maximului funcției f_0 . Funcția

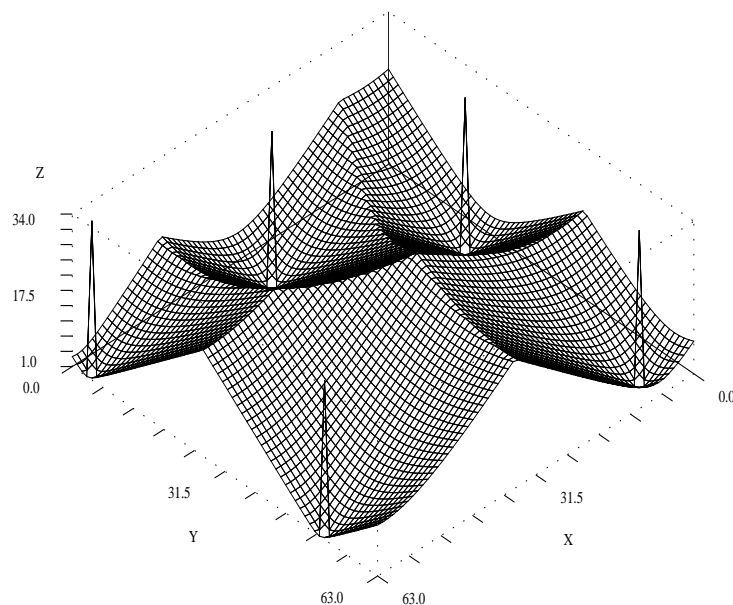


Figura B.5: Graficul unei funcții deceptive

are dezavantajul că este discontinuă, dar se poate imagina cu ușurință o funcție continuă care să aibă un astfel de relief. O astfel de funcție se numește deceptivă deoarece un algoritm care încearcă să optimizeze așa ceva, cu greu poate găsi cele k maxime globale.

B.2 Probleme de optimizare cu restricții

1. Să se minimizeze funcția

$$G1(\mathbf{x}, \mathbf{y}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5 \sum_{i=1}^4 x_i^2 - \sum_{i=1}^9 y_i,$$

cu restricțiile:

$$\begin{aligned}
 2x_1 + 2x_2 + y_6 + y_7 &\leq 10, & 2x_1 + 2x_3 + y_6 + y_8 &\leq 10, \\
 2x_2 + 2x_3 + y_7 + y_8 &\leq 10, & -8x_1 + y_6 &\leq 0, \\
 -8x_2 + y_7 &\leq 0, & -8x_3 + y_8 &\leq 0, \\
 -2x_4 - y_1 + y_6 &\leq 0, & -2y_2 - y_3 + y_7 &\leq 0, \\
 -2y_4 - y_5 + y_8 &\leq 0, & 0 \leq x_i \leq 1, i = 1, 2, 3, 4, \\
 0 \leq y_i \leq 1, i = 1, 2, 3, 4, 5, 9, & & 0 \leq y_i, i = 6, 7, 8.
 \end{aligned}$$

Funcția are un minim global egal cu -15 în punctul $(\mathbf{x}, \mathbf{y}) = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$.

2. Să se minimizeze funcția

$$G2(\mathbf{x}) = x_1 + x_2 + x_3,$$

cu restricțiile:

$$\begin{aligned}
 1 - 0.0025(x_4 + x_6) &\geq 0, \\
 x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 &\geq 0, \\
 1 - 0.0025(x_5 + x_7 - x_4) &\geq 0, \\
 x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 &\geq 0, \\
 1 - 0.01(x_8 - x_5) &\geq 0, \\
 x_3x_8 - 1250000 - x_3x_5 + 2500x_5 &\geq 0, \\
 100 \leq x_1 \leq 10000, \\
 1000 \leq x_i \leq 10000, \quad i = 2, 3, \\
 10 \leq x_i \leq 1000, \quad i = 4, \dots, 8.
 \end{aligned}$$

Problema are 3 restricții liniare și 3 neliniare; funcția $G2$ este liniară și are un minim global egal cu 7049.330923 în punctul $\mathbf{x} = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$. Toate cele șase restricții sunt active în optimul global.

3. Să se minimizeze funcția

$$\begin{aligned}
 G3(\mathbf{x}) &= (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + \\
 &+ x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,
 \end{aligned}$$

cu restricțiile:

$$\begin{aligned}
 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 &\geq 0, \\
 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 &\geq 0, \\
 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 &\geq 0, \\
 -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 &\geq 0, \\
 -10.0 \leq x_i \leq 10.0, \quad i = 1, \dots, 7.
 \end{aligned}$$

Problema are 4 restricții neliniare; funcția $G3$ este neliniară și are un minim global egal cu 680.6300573 în punctul (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227). Două din cele patru restricții (prima și ultima) sunt active în optimul global.

4. Să se minimizeze funcția

$$G4(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5},$$

cu restricțiile:

$$\begin{aligned} x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 &= 10, \\ x_2 x_3 - 5 x_4 x_5 &= 0, \\ x_1^3 + x_2^3 &= -1, \\ -2.3 \leq x_i &\leq 2.3, \quad i = 1, 2, \\ -3.2 \leq x_i &\leq 3.2, \quad i = 3, 4, 5. \end{aligned}$$

Problema are 3 restricții neliniare de tip egalitate; funcția $G4$ este neliniară și are un minim global egal cu 0.0539498478 în punctul (-1.717143, 1.595709, 1.827247, -0.763641, -0.7636450).

5. Să se minimizeze funcția

$$\begin{aligned} G5(\mathbf{x}) &= x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + \\ &+ 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45, \end{aligned}$$

cu restricțiile:

$$\begin{aligned} 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 &\geq 0, \\ -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 &\geq 0, \\ -10x_1 + 8x_2 + 17x_7 - 2x_8 &\geq 0, \\ -x_1^2 - 2(x_2 - 2)^2 + 2x_1 x_2 - 14x_5 + 6x_6 &\geq 0, \\ 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 &\geq 0, \\ -5x_1 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 &\geq 0, \\ 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} &\geq 0, \\ -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 &\geq 0, \\ -10.0 \leq x_i &\leq 10.0, \quad i = 1, \dots, 10. \end{aligned}$$

Problema are 3 restricții liniare și 5 neliniare; funcția $G5$ este pătratică și are un minim global egal cu 24.3062091 în punctul (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927). Șase din cele opt restricții (toate cu excepția ultimilor două) sunt active în optimul global.

6. Să se minimizeze funcția

$$G6(\mathbf{x}, y) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5 \sum_{i=1}^5 x_i^2,$$

cu restricțiile:

$$\begin{aligned} 6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 &\leq 6.5, \\ 10x_1 + 10x_3 + y &\leq 20, \\ 0 \leq x_i \leq 1, \quad i &= 1, \dots, 5, \\ 0 \leq y. \end{aligned}$$

Funcția are un minim global egal cu -213 în punctul $(0, 1, 0, 1, 1, 20)$.

7. Să se maximizeze funcția

$$G7(\mathbf{x}) = \frac{3x_1 + x_2 - 2x_3 + 0.8}{2x_1 - x_2 + x_3} + \frac{4x_1 - 2x_2 + x_3}{7x_1 + 3x_2 - x_3},$$

cu restricțiile:

$$\begin{aligned} x_1 + x_2 - x_3 &\leq 1, \\ -x_1 + x_2 - x_3 &\leq -1, \\ 12x_1 + 5x_2 + 12x_3 &\leq 34.8, \\ 12x_1 + 12x_2 + 7x_3 &\leq 29.1, \\ -6x_1 + x_2 + x_3 &\leq -4.1, \\ 0 \leq x_i, \quad i &= 1, 2, 3. \end{aligned}$$

Funcția are un maxim global egal cu 2.471428 în punctul $(1, 0, 0)$.

8. Să se minimizeze funcția

$$G8(x, \mathbf{y}) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5,$$

cu restricțiile:

$$\begin{aligned} x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 &\leq 16, \\ -8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 &\leq -1, \\ 2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 &\leq 24, \\ 0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 &\leq 12, \\ -0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 &\leq 3, \\ y_3 \leq 1, \quad y_4 \leq 1 \quad \text{și} \quad y_5 &\leq 2, \\ x \geq 0, \quad y_i \geq 0 \quad \text{pentru} \quad 1 \leq i &\leq 5. \end{aligned}$$

Funcția are un minim global egal cu -11 în punctul $(0, 6, 0, 1, 1, 0)$.

Anexa C

Exemple de probleme de optimizare a dispozitivelor electromagnetice

Această anexă prezintă două probleme legate de optimizarea dispozitivelor electromagnetice. Prima aplicație reprezintă optimizarea unui dispozitiv de stocare a energiei magnetice, iar a doua optimizarea unei matrițe folosită pentru orientarea pulberilor în câmp magnetic. Ambele probleme sunt probleme de test¹ propuse de comunitatea internațională în cadrul “TEAM² Workshop”. Descrierea problemelor poate fi găsită pe Internet [9]. Rezultate privind optimizarea acestor două probleme se găsesc în teza [17], al cărei rezumat se găsește la adresa <http://www.lmn.pub.ro/~gabriela>.

C.1 Dispozitiv de stocare a energiei magnetice

Problema TEAM Workshop 22 constă în optimizarea unui dispozitiv SMES³. Dispozitivele SMES sunt dispozitive care stochează energia în câmpuri magnetice. În principiu ele sunt construite din bobine realizate din materiale supraconductoare. Bobinele sunt alimentate printr-un comutator de la un convertizor de putere, după care comutatorul se deschide simultan cu scurtcircuitarea bornelor bobinelor. Curentul circulă în bobine fără a scădea în timp datorită rezistenței nule a supraconductoarelor. Astfel de dispozitive pot fi folosite pentru stabilizarea fluctuațiilor de putere în sistemele energetice.

Există două tipuri diferite de bobine folosite în dispozitivele SMES: solenoizii și toroizii. Tehnica de realizare a unei bobine solenoidale este foarte simplă, în timp ce

¹”Benchmark problems”

²TEAM (Testing Electromagnetic Analysis Models) reprezintă un grup internațional de lucru constituit în scopul comparării diferitelor programe folosite la analiza câmpului electromagnetic.

³Superconducting Magnetic Energy Storage

realizarea unei bobine toroidale este mult mai sofisticată și necesită o cantitate de material supraconductor aproape de două ori mai mare. Avantajul unei bobine toroidale constă în faptul că, datorită geometriei sale, câmpul magnetic în spațiul înconjurător este practic nul. Acest rezultat este valabil în cazul în care bobina toroidală este înfășurată perfect, lucru care nu se realizează în practică deoarece toroizii sunt construiți din mai mulți solenoizi plasați pe o formă de tor. Cu toate acestea, un astfel de toroid produce un câmp magnetic la mare depărtare de el mult mai mic decât câmpul magnetic produs de un singur solenoid.

Problema TEAM 22 constă într-o configurație SMES care are doi solenoizi prin care trec curenți de sensuri opuse. În acest fel câmpul de dispersie în cazul folosirii a doi solenoizi este mai mic decât câmpul de dispersie al unui singur solenoid. Această construcție simulează câmpul magnetic al unui quadripol care scade (la depărtare) cu puterea a 5-a a razei, spre deosebire de câmpul magnetic al unui solenoid (un dipol magnetic) care scade la depărtare cu puterea a 3-a a razei. Desigur, această construcție consumă mai mult material decât un singur solenoid, avantajul economiei de material (fața de cazul toroidului) nemaifiind semnificativ. Totuși, construcția cu solenoizi este mult mai simplă din punct de vedere tehnologic.

Definiția problemei este prezentată în cele ce urmează.

Un dispozitiv SMES (figura C.1) trebuie optimizat astfel încât să fie atinse următoarele obiective:

- Energia magnetică stocată în dispozitiv să fie 180 MJ;
- Câmpul magnetic trebuie să satisfacă condiția fizică ce garantează supraconductibilitatea;
- Câmpul de dispersie (măsurat la o distanță de 10 metri de dispozitiv) să fie cât mai mic posibil.

Problema are 8 parametri ($R_1, R_2, h_1/2, h_2/2, d_1, d_2, J_1, J_2$) ale căror restricții sunt prezentate în tabelul C.1.

	R_1 [m]	R_2 [m]	$h_1/2$ [m]	$h_2/2$ [m]	d_1 [m]	d_2 [m]	J_1 [MA/m ²]	J_2 [MA/m ²]
min	1.0	1.8	0.1	0.1	0.1	0.1	10.0	-30.0
max	4.0	5.0	1.8	1.8	0.8	0.8	30.0	-10.0

Tabelul C.1: Restricții de domeniu ale variabilelor de optimizare

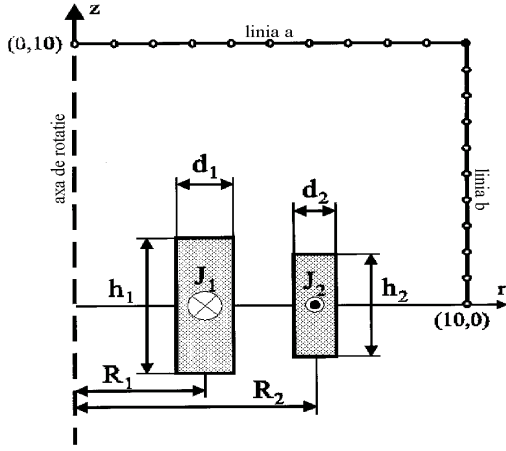


Figura C.1: Dispozitiv SMES cu doi solenoizi

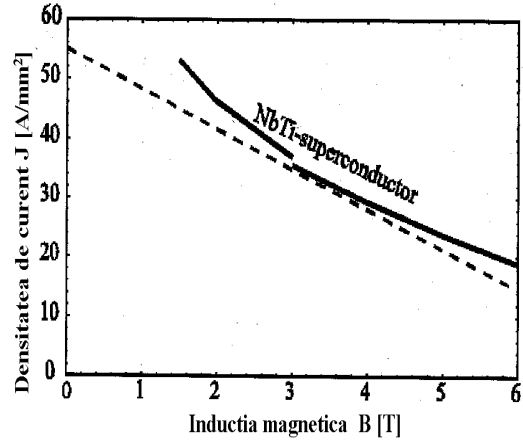


Figura C.2: Restricția impusă pentru supraconductor

Condiția care asigură faptul că bobinele nu își pierd starea supraconductoare (“quench condition”) constă într-o relație între modulul densității de curent și valoarea maximă a modulului inducției magnetice $|B|$ în bobine, așa cum arată figura C.2. Ecuația (C.1) este o aproximare a curbei din figura C.2,

$$|J| = (-6.4|B| + 54.0) \text{ A/mm}^2. \quad (\text{C.1})$$

Funcția obiectiv propusă este

$$F = \frac{B_{\text{stray}}^2}{B_{\text{norm}}^2} + \frac{|E - E_{\text{ref}}|}{E_{\text{ref}}}, \quad (\text{C.2})$$

unde $E_{\text{ref}} = 180 \text{ MJ}$, $B_{\text{norm}} = 2.0 \cdot 10^{-4} \text{ T}$ și

$$B_{\text{stray}}^2 = \frac{\sum_{i=1}^{22} |B_{\text{stray}_i}|^2}{22}. \quad (\text{C.3})$$

Valoarea B_{stray}^2 este obținută după evaluarea câmpului în 22 de puncte echidistante de pe liniile a și b (figura C.1).

C.2 Matriță cu electromagnet

Problema TEAM 25 constă în optimizarea formei unei matrițe cu electromagnet, folosită pentru orientarea pulberilor magnetice în procesul de sinterizare a pieselor polare pentru micromașini. Matrița și electromagnetul sunt confecționate din oțel, forma matriței fiind

astfel realizată încât să se genereze un câmp magnetic radial într-o cavitate ce va fi umplută cu pulberea magnetică.

Figura C.3 reprezintă o secțiune transversală a matriței cu electromagnet, iar figura C.4 prezintă un detaliu în zona de interes. Matrița trebuie optimizată pentru două valori ale solenațiilor bobinelor: 4253 amperi-spiră și respectiv 17500 amperi-spiră astfel încât câmpul magnetic în cavitate (de-a lungul curbei e-f din figura C.4) să fie orientat radial și să aibă următoarele valori:

- în cazul solenației mici (4253 amperi-spiră)

$$B_x = 0.35 \cos \theta \text{ [T]} \quad B_y = 0.35 \sin \theta \text{ [T]} \quad (\text{C.4})$$

- în cazul solenației mari (17500 amperi-spiră)

$$B_x = 1.5 \cos \theta \text{ [T]} \quad B_y = 1.5 \sin \theta \text{ [T]} \quad (\text{C.5})$$

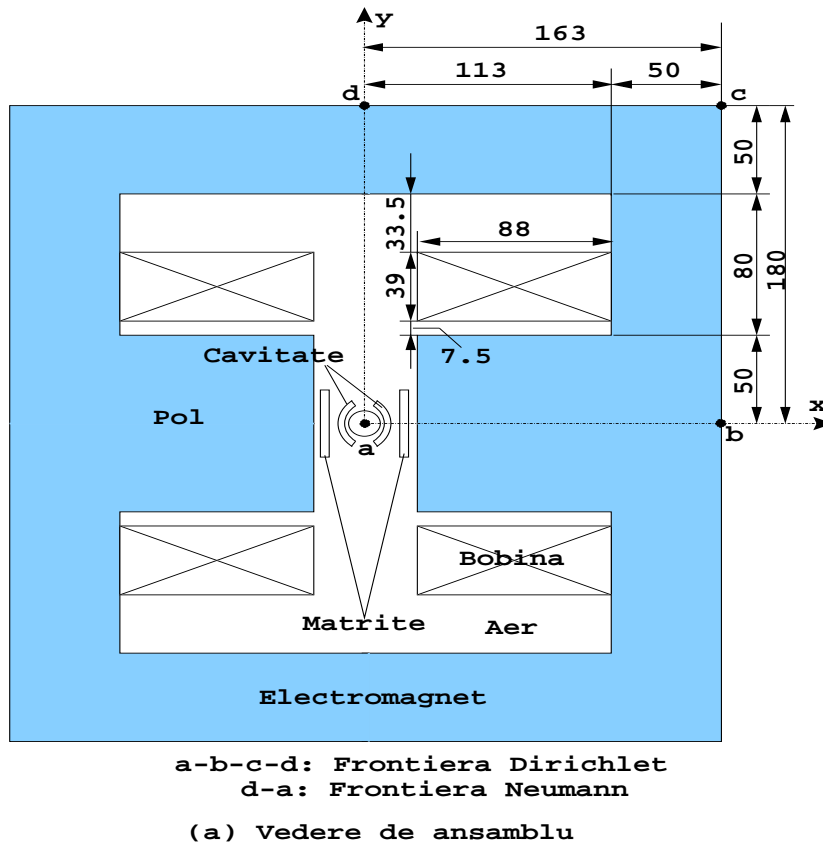


Figura C.3: Matriță cu electromagnet

Analizele preliminare făcute la curenți mici arată că distribuția de câmp cerută poate fi obținută (cu eroare acceptabilă) adoptând pentru matriță o formă obținută printr-o

combinație de linie dreaptă, cerc și elipsă. Matrița este alcătuită din două părți, numite forme. Forma interioară a matriței este presupusă a fi un cerc de rază R_1 . Partea dinspre interior a forme exterioare este reprezentată de o elipsă de semiaxe L_2 și L_3 și o linie paralelă cu axa x, ca în figura C.4.

Variabilele de proiectare sunt: R_1 - raza forme interioare, L_2 - axa lungă a elipsei, L_3 - axa scurtă a elipsei, L_4 - lungimea pintenului forme exterioare. Proiectatul are libertatea să aleagă și alte parametrizări ale curbei g-h a forme interioare și ale curbei i-j-k-m corespunzătoare forme exterioare.

Se constată că la curenți mari, dacă forma interioară și cea exterioară sunt reprezentate prin cerc și elipsă rezultatele nu sunt satisfăcătoare.

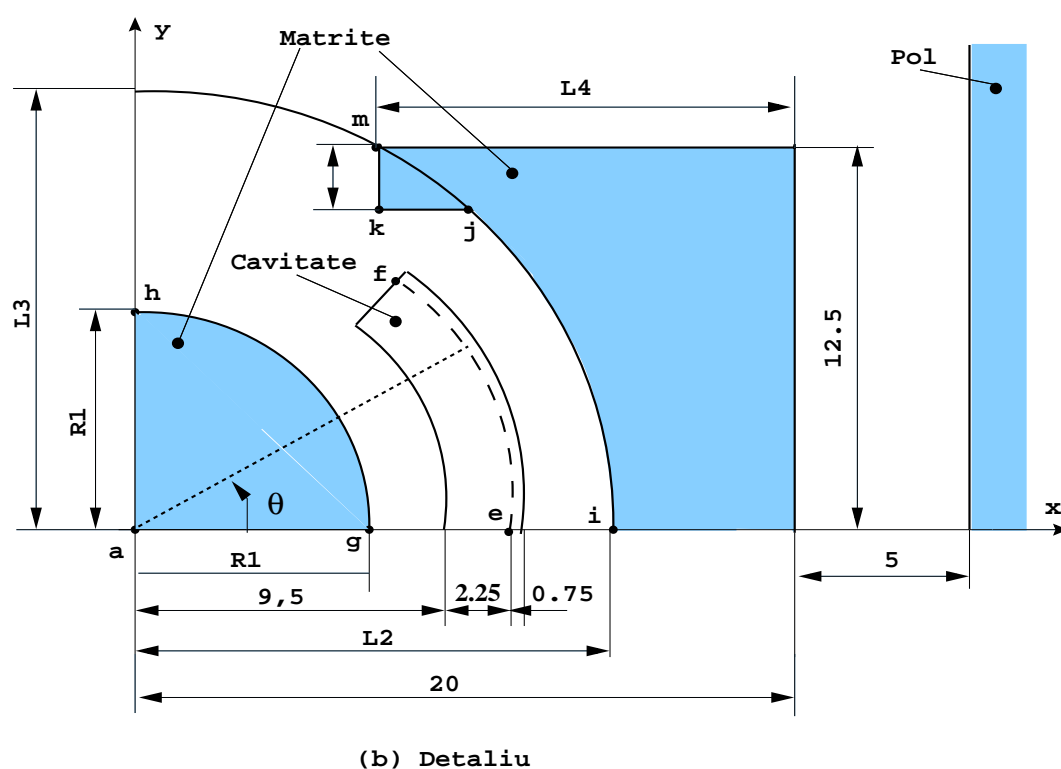


Figura C.4: Detaliu în zona de interes

Matrița și electromagnetul sunt din oțel, având curba de magnetizare prezentată în figura C.5. Punctele acestei curbe sunt cele din tabelul C.2.

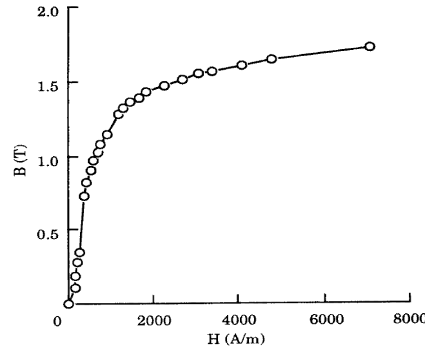


Figura C.5: Curba de magnetizare a oțelului

B [T]	0.0	0.11	0.18	0.28	0.35	0.74	0.82	0.91
H [A/m]	0.0	140	178	215	253	391	452	529
B [T]	0.98	1.02	1.08	1.15	1.27	1.32	1.36	1.39
H [A/m]	596	677	774	902	1164	1299	1462	1640
B [T]	1.42	1.47	1.51	1.54	1.56	1.60	1.64	1.72
H [A/m]	1851	2262	2685	3038	3395	4094	4756	7079

Tabelul C.2: Punctele curbei de magnetizare

	R_1 [mm]	L_2 [mm]	L_3 [mm]	L_4 [mm]
min	5	12.6	14	4
max	9.4	18	45	19

Tabelul C.3: Restricții de domeniu ale variabilelor de optimizare

Problema are 4 parametri (R_1, L_2, L_3, L_4) care pot varia continuu între limitele prezentate în tabelul C.3.

Funcția obiectiv propusă este

$$F = \sum_{i=1}^n [(B_{xp_i} - B_{xo_i})^2 + (B_{yp_i} - B_{yo_i})^2]. \quad (C.6)$$

Indicii p și o se referă la valorile calculate, respectiv la valorile specificate. Se consideră $n = 10$ puncte situate pe curba e-f (un arc de cerc de rază 11.75 mm, conform figurii C.4), în punctele corespunzătoare unghiurilor de: 0, 5, 10, 15, 20, 25, 30, 35, 40 și respectiv 45 de grade.

Se cere de asemenea calculul erorii maxime ε_{Bmax} a modulului și eroarea maximă $\varepsilon_{\theta max}$ a unghiului inducției magnetice, definite astfel:

$$\varepsilon_{Bmax} = \max_{1 \leq i \leq n} \left| \frac{B_{pi} - B_{oi}}{B_{oi}} \right|, \quad (C.7)$$

$$\varepsilon_{\theta max} = \max_{1 \leq i \leq n} |\theta_{Bpi} - \theta_{Boi}|. \quad (C.8)$$

Autorii acestei probleme au realizat experimental două astfel de matrițe, pentru care au măsurat valorile inducției în punctele specificate. Aceste rezultate experimentale sunt prezentate în [60].

Anexa D

Metode deterministe pentru optimizarea locală

D.1 Metode de optimizare deterministe pentru probleme fără restricții

Cunoașterea metodelor de optimizare fără restricții este foarte importantă pentru că de multe ori o problemă de optimizare cu restricții este redusă la o problemă de optimizare fără restricții. Iată o clasificare a metodelor de optimizare pentru probleme fără restricții [16, 30, 53].

- **Optimizare unidimensională**

O metodă *fără calculul derivatei* trebuie să încadreze mai întâi minimumul. O astfel de metodă de încadrare este **metoda secțiunii de aur** care găsește un triplet de puncte $a < b < c$ astfel încât $f(b)$ este mai mic și decât $f(a)$ și decât $f(c)$. Dacă funcția de optimizat are caracteristici suplimentare (de exemplu derivata de ordin doi este continuă, atunci încadrarea obținută din metoda secțiunii de aur se poate rafina cu ajutorul unei interpolări parabolice. O astfel de metodă este **metoda Brent**.

Pentru minimizări unidimensionale *cu calculul derivatei* se poate folosi de exemplu o variantă a metodei Brent care utilizează și informații despre prima derivată.

- **Optimizarea multidimensională**

În acest caz trebuie ales între metode care au un necesar de memorie de ordinul N^2 (N reprezintă numărul de dimensiuni) sau de ordinul N . Pentru valori mici ale lui N această cerință nu este o restricție.

Una dintre metodele *fără calculul derivatei* este **metoda simplexului descendent**,

datorată lui Nelder și Mead. Ea nu trebuie confundată cu metoda simplex din programarea liniară. Această metodă este foarte încet convergentă, dar este foarte robustă și nu face presupuneri speciale despre funcție. Un simplex în N dimensiuni este un poligon convex având $N + 1$ vârfuri (triunghi pentru $N = 2$, tetraedru pentru $N = 3$). În cazul bidimensional, vârful unui triunghi caracterizat de valoarea cea mai mare a funcției obiectiv este reflectat în raport cu linia care trece prin celelalte două vârfuri. Valoarea funcției în acest punct nou este comparată cu valorile rămase în celelalte două puncte. În funcție de acest test noul punct poate fi acceptat sau rejectat. Când nu se mai realizează nici o îmbunătățire laturile triunghiului sunt contractate și procedura se repetă. În acest fel este posibil ca vârfurile triunghiului să convergeze către un punct în care funcția de minimizat este minimă. Necesarul de memorie este N^2 .

O altă metodă este metoda alegerii direcțiilor, **metoda Powell** fiind prototipul acestei categorii. Această metodă cere o minimizare unidimensională (de exemplu metoda Brent), iar necesarul de memorie este N^2 .

În ce privește metodele *cu calculul derivatei*, există două familii mari de algoritmi pentru minimizarea multidimensională, care necesită calcul de derivate. Ambele familii cer un subalgoritm de minimizare unidimensional care poate sau nu să folosească informații legate de derivată. Prima familie (de ordinul unu) cuprinde **metoda pașilor descendenți** și metode de **gradienti conjugați**. Metoda pașilor descendenți este una din cele mai populare și mai vechi metode folosite în optimizare. Soluția optimă este obținută printr-un număr de iterații consecutive, în fiecare din ele o soluție nouă fiind obținută printr-o deplasare plecând din punctul vechi pe o anumită direcție. Direcția este opusă direcției gradientului iar lungimea (pasul) este o mărime constantă sau nu. Există mai multe variante de algoritmi pentru metoda gradientilor conjugați, de exemplu algoritmul Fletcher-Reeves sau algoritmul Polak-Ribiere (ceva mai bun). Necesarul de memorie este de ordinul N .

A doua familie (de ordinul doi) cuprinde **metoda Gauss-Newton** și **metode quasi-Newton**, de exemplu algoritmul Davidon-Fletcher-Powell, sau algoritmul Broyden-Fletcher-Goldfarb-Shanno. Într-o metodă quasi-Newton direcția de căutare $\Delta \mathbf{x}$ la iterația q este definită ca $\Delta \mathbf{x} = -\mathbf{H}^{-1} \text{grad } f(\mathbf{x})$ unde \mathbf{H} este o aproximare a matricei Hessian a funcției obiectiv F . La primul pas \mathbf{H} este luată matricea unitate (primul pas este deci identic cu cel al metodei pașilor descendenți); apoi \mathbf{H} este recalculat succesiv în concordanță cu o procedură de metrică variabilă. Necesarul de memorie este de N^2 . În metoda Gauss-Newton se folosește chiar matricea Hessian.

Pe scurt, o minimizare multidimensională se reduce la un șir de iterații de tipul: $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k$ unde \mathbf{d}^k este direcția de căutare la iterația k iar α^k este un coeficient determinat printr-o metodă de minimizare unidimensională. Metodele diferă prin alegerea lui \mathbf{d}^k . Tabelul 1 cuprinde valorile acestor direcții pentru metodele de ordin superior:

Metodă	\mathbf{d}^k	Observații
Pașilor descendenți	$-\nabla^k f$	
Gradientilor conjugați	$-\nabla^k f + \mathbf{d}^{k-1} \frac{(\nabla^k f)^T \nabla^k f}{(\nabla^{k-1} f)^T \nabla^{k-1} f}$	
Gauss-Newton	$-(\mathbf{G}^k)^{-1} \nabla^k f$	\mathbf{G} este matricea Hessian: $\mathbf{G} = 2\mathbf{J}^T \mathbf{J}$ unde \mathbf{J} este matricea Jacobi
quasi-Newton	$-(\mathbf{H}^k)^{-1} \nabla^k f$	\mathbf{H} este o aproximare a matricei Hessian

Tabelul D.1: Direcții de căutare într-o metodă de optimizare de ordin superior

D.2 Tratarea restricțiilor

În ceea ce privește problemele cu restricții, metoda simplex a fost prima dezvoltată special pentru o astfel de problemă [53]. Această metodă se bazează pe faptul că o funcție obiectiv liniară își atinge extremele pe frontieră și atunci căutarea lor se face numai pe frontieră. Mai există o clasă de metode numite ”metode de punct interior” [47], mai rar folosite decât metoda simplex.

Aceste două metode rezolvă însă doar problema programării liniare adică aceea în care atât funcția de minimizat cât și restricțiile sunt liniare, situație extrem de rar întâlnită în optimizarea dispozitivelor electromagnetice. De aceea o problemă foarte importantă o constituie tratarea restricțiilor. Există mai multe posibilități de a elimina restricțiile.

• Folosirea funcției Lagrange [16, 30, 47]

Această abordare nu folosește funcția propriu-zisă f ci o nouă funcție numită funcție Lagrange, fiind deci o metodă de transformare.

Pentru concizie vom nota cu $f : \mathbb{R}^n \rightarrow \mathbb{R}$ funcția obiectiv (reală), cu $\mathbf{x} \in \mathbb{R}^n$ vectorul variabilelor, restricțiile sunt reprezentate de funcția $c : \mathbb{R}^n \rightarrow \mathbb{R}^p$ unde unele componente ale lui c sunt restricții de tip inegalitate, de forma $c_i(\mathbf{x}) \leq 0$ pentru i într-o mulțime de indici notată \mathcal{I} , alte componente ale lui c sunt restricții de tip egalitate de tip $c_i(\mathbf{x}) = 0$ pentru indici i dintr-o mulțime de indici notată \mathcal{E} . Problema se poate formula astfel

$$\min \{f(\mathbf{x}) | c_i(\mathbf{x}) \leq 0, i \in \mathcal{I}, c_i(\mathbf{x}) = 0, i \in \mathcal{E}\}, \quad (\text{D.1})$$

unde $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$, \mathcal{I} și \mathcal{E} sunt mulțimile de indici corespunzătoare inegalităților și egalităților ¹.

¹Inegalitățile de domeniu sunt incluse în $c_i(\mathbf{x}) \leq 0, i \in \mathcal{I}$.

atinge în punctul 3 (în care restricțiile active sunt g_2 și g_3). Mutarea restricției g_2 în g_2^* în domeniul admisibil nu modifică mai puțin valoarea minimului (noul minim este punctul 4), decât cazul în care se mută restricția g_3 în g_3^* (minimul în acest caz este punctul 5). În punctul optim (punctul 3) $-\nabla f = \lambda_2 \nabla g_2 + \lambda_3 \nabla g_3$ unde $\lambda_2 > \lambda_3$

Condiția de ordinul doi (existență și stabilitate) - cere ca $(\mathbf{x}^*, \lambda^*)$ să satisfacă condiția de ordinul unu $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \lambda^*) = 0$ și, în plus, Hessianul funcției Lagrange

$$\nabla_{\mathbf{xx}}^2 \mathcal{L}(\mathbf{x}^*, \lambda^*) = \nabla^2 f(\mathbf{x}^*) + \sum_{i \in \mathcal{A}^*} \lambda_i^* \nabla^2 c_i(\mathbf{x}^*),$$

să satisfacă $\mathbf{w}^T \nabla_{\mathbf{xx}}^2 \mathcal{L}(\mathbf{x}^*, \lambda^*) \mathbf{w} > 0$ pentru vectorii nenuli \mathbf{w} din mulțimea

$$\{\mathbf{w} \in \mathbb{R}^n | \nabla c_i(\mathbf{x}^*)^T \mathbf{w} = 0, i \in \mathcal{I}_+^* \cup \mathcal{E}, \nabla c_i(\mathbf{x}^*)^T \mathbf{w} \leq 0, i \in \mathcal{I}_0^*\},$$

unde

$$\mathcal{I}_+^* = \{i \in \mathcal{A}^* \cap \mathcal{I} | \lambda_i^* > 0\}, \quad \mathcal{I}_0^* = \{i \in \mathcal{A}^* \cap \mathcal{I} | \lambda_i^* = 0\}.$$

Această condiție garantează că problema de optimizare are o comportare stabilă în jurul extremului.

În funcție de cum este folosită mai departe această funcție Lagrange (cum sunt calculați multiplicatorii Lagrange și cum este minimizată funcția Lagrange), metodele sunt: metode de gradient redus, metode de programare pătratică și liniară, metode bazate pe funcții Lagrange modificate² și funcții de penalizare. Descrierea detaliată a metodelor se găsește în [47].

O altă metodă care folosește funcția Lagrange este metoda asimptotelor³ [57]. În această metodă ideea de bază este de a se înlocui problema inițială cu o secvență de subprobleme convexe care au o formă simplă. Pentru fiecare subproblemă se definește Lagrangeanul $L(\mathbf{x}, \lambda) = F(\mathbf{x}) + \sum \lambda_i F_i(\mathbf{x})$ unde F și F_i sunt aproximații convexe ale funcției obiectiv și restricțiilor, suma se face după indicii i aparținând restricțiilor active, și se rezolvă problema duală $\max_{\lambda \geq 0} \{\min_{\mathbf{x}} L(\mathbf{x}, \lambda)\}$. Aceeași metodă este descrisă și folosită în [56].

• **Minimizări secvențiale pentru funcții obiectiv care includ restricțiile penalizate**⁴ [30, 57]

– *Eliminarea restricțiilor de tip egalitate după metoda lui Courant*

Există situații când tehnica Lagrange este dificil de aplicat. În aceste situații, în practică sunt preferate alte metode. Una din cele mai cunoscute tehnici alternative de

²Se întâlnește în literatură cu prescurtarea ALM - "Augmented Lagrange Method".

³"Method of Moving Asymptotes"

⁴"Penalty Methods"

tratare a restricțiilor a fost inițiată de R.Courant [30]. În această metodă se minimizează funcția

$$F(x_1, \dots, x_n) = f(x_1, \dots, x_n) + r \sum_{i=1}^p [h_i(x_1, \dots, x_n)]^2 \quad r > 0 \quad (\text{D.4})$$

ca o problemă fără restricții, pentru o secvență de valori crescătoare ale lui r . Funcția f este astfel "penalizată" atunci când restricțiile nu sunt satisfăcute. Când r tinde către infinit, suma pătratelor restricțiilor este forțată să tindă spre zero. În acest fel, șirul minimelor succesive ale funcției F tind către soluția problemei inițiale.

– *Eliminarea restricțiilor de tip inegalitate*

Să presupunem că problema are m restricții neliniare de tip inegalitate

$$g_i(x_1, \dots, x_n) \geq 0 \quad i = 1, \dots, m. \quad (\text{D.5})$$

Există două abordări posibile: una se numește "the boundary-following approaches" (abordări care urmăresc frontiera) și cealaltă este "penalty-function techniques" (tehnici bazate pe funcții de penalizare).

După cum le sugerează numele, abordările care urmăresc frontiera sugerează ca atunci când o restricție este (sau aproape este) violată, se urmărește frontiera domeniului variabilelor, corespunzătoare acelei restricții, până se găsește un punct satisfăcător. Dacă frontiera este puternic neliniară, convergența unor astfel de abordări este înceată. În aceste cazuri se folosește tehnica funcțiilor de penalizare. Să considerăm de exemplu funcțiile:

$$\begin{aligned} F_1(x_1, \dots, x_n) &= f(x_1, \dots, x_n) + r \sum_{i=1}^m \frac{1}{g_i(x_1, \dots, x_n)} \quad r > 0, r \rightarrow 0, \\ F_2(x_1, \dots, x_n) &= f(x_1, \dots, x_n) - r \sum_{i=1}^m \ln[g_i(x_1, \dots, x_n)] \quad r \rightarrow 0, \\ F_3(x_1, \dots, x_n) &= f(x_1, \dots, x_n) + r \sum_{i=1}^m (\min(0, g_i(x_1, \dots, x_n)))^2 \quad r \rightarrow \infty. \end{aligned}$$

Dacă aceste funcții sunt minimizeate secvențial pentru un șir de valori pozitive ale lui r (monoton descrescător pentru F_1 și F_2 și monoton crescător pentru F_3), atunci șirul minimelor funcțiilor cu penalizare (fără restricții) tinde către minimul funcției originale.

O altă metodă de eliminare a restricțiilor de tip inegalitate $g_i(\mathbf{x}) \leq 0$ este următoarea [57]. Se folosește o funcție

$$W(\mathbf{x}, r) = f(\mathbf{x}) + r \sum_{i=1}^m F_i(g_i(\mathbf{x})),$$

unde r este un parametru de penalizare iar funcția F_i asociată restricției g_i este definită astfel

$$F_i(g_i(\mathbf{x})) = \begin{cases} -\frac{1}{g_i(\mathbf{x})} & \text{pentru } g_i(\mathbf{x}) \leq \varepsilon \\ \frac{1}{\varepsilon} \left[3\frac{g_i(\mathbf{x})}{\varepsilon} - \left(\frac{f_i(\mathbf{x})}{\varepsilon} \right)^2 - 3 \right] & \text{pentru } g_i(\mathbf{x}) > \varepsilon \end{cases},$$

unde ε este un parametru. Soluția problemei de optimizare este obținută prin minimizări succesive ale lui W pentru o secvență de valori descrescătoare a lui r . Aceeași metodă este folosită și în [35, 38].

D.3 Optimizare neliniară fără restricții

Problema optimizării fără restricții joacă un rol central în programele de optimizare deoarece problemele cu restricții sunt reduse în cele mai multe cazuri la această problemă. Problema optimizării fără restricții:

$$\min\{f(x) : x \in \mathbb{R}^n\},$$

constă în căutarea unui minim local $x^* \in \mathbb{R}^n$, astfel încât $f(x^*) \leq f(x)$ pentru orice x într-o vecinătate a punctului x^* .

Metoda Newton de rezolvare a acestei probleme stă la baza unei largi și importante clase de algoritmi. Ea se bazează pe minimizarea iterativă a unor modele pătratice pentru funcția obiectiv f :

$$q_k(s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T \nabla^2 f(x_k) s,$$

în care

$$\nabla f(x) = [\partial_1 f(x), \dots, \partial_n f(x)]^T$$

este vectorul gradient, iar

$$\nabla^2 f(x) = [\partial_j \partial_i f(x)]$$

este matricea Hessian a funcției f . Dacă matricea Hessian este pozitiv definită, atunci $q_k(s)$ are un minim unic, care se obține prin rezolvarea sistemului de ecuații liniare cu matrice simetrică de dimensiune $n \times n$:

$$\nabla^2 f(x_k) s_k = -\nabla f(x_k).$$

Corecția s_k astfel calculată este folosită la determinarea următoarei iterații:

$$x_{k+1} = x_k + s_k.$$

Cu toate că evaluarea matricei Hessian poate necesita efort mare de calcul, în destul de multe probleme acest efort este justificat. Convergența șirului de iterații x_0, x_1, x_2, \dots

este garantată, dacă inițializarea x_0 este suficient de apropiată de minimul local x^* și pentru care $\nabla^2 f(x^*)$ este pozitiv definită. În acest caz, metoda Newton are o convergență pătratică, existnd o constantă β , astfel înct

$$\|x_{k+1} - x^*\| \leq \beta \|x_k - x^*\|^2.$$

Dacă inițializarea este îndepărtată de soluție, atunci este posibil ca matricea Hessian să nu fie pozitiv definită, deci problema minimizării modelului pătratic să nu aibă soluție. Pentru a evita astfel de situații este necesară modificarea metodei Newton. Cele mai des utilizate modificări care urmăresc extinderea domeniului de convergență al metodei Newton sunt cele de “**căutare liniară**” (line search) și “**regiunea de încredere**” (trust region).

Metoda căutării liniare se bazează pe iterațiile:

$$x_{k+1} = x_k + \alpha_k d_k,$$

în care d_k este direcția de căutare, iar $\alpha_k > 0$ este ales astfel înct $f(x_{k+1}) < f(x_k)$. Direcțiile de căutare d_k sunt determinate prin rezolvarea sistemului liniar:

$$(\nabla^2 f(x_k) + E_k)d_k = -\nabla f(x_k),$$

obținut prin perturbarea modelului pătratic, prin adăugarea la diagonală matricei Hessian a perturbării E_k , care face ca aceasta să devină pozitiv definită. Determinarea valorii scalarului α_k se face prin căutare unidimensională, după direcția d_k a minimului funcției $\varphi(x) = f(x_k + \alpha d_k)$, de obicei prin interpolare pătratică sau cubică. Valoarea α_k se consideră acceptabilă dacă sunt satisfăcute condițiile de “**scădere suficientă**”:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \mu \alpha_k \nabla f(x_k)^T d_k,$$

respectiv de “**curbură**”:

$$|\nabla f(x_k + \alpha_k d_k)^T d_k| \leq \eta |\nabla f(x_k)^T d_k|,$$

în care se folosesc de obicei valorile $\mu = 10^{-3}$, $\eta = 0.9$. Prima condiție asigură scăderea funcției obiectiv, iar a doua condiție asigură faptul că α_k determină o apropiere suficientă de minimul funcției $\varphi(\alpha)$.

Metoda regiunii de încredere se bazează pe observația că modelul pătratic $q_k(s)$ este o bună aproximare a funcției $f(x_k + s)$ doar pentru s suficient de mic. De aceea este rezonabil să se aleagă pasul s_k ce reprezintă soluția subproblemei de minimizare cu restricții:

$$\min \{q_k(s) : \|D_k s\|_2 \leq \Delta_k\},$$

în care $\Delta_k > 0$ este un parametru real ce descrie mărimea regiunii de încredere, iar D_k este o matrice diagonală de scalare. Mărimea regiunii de încredere este ajustată la fiecare

iterație, în funcție de abaterea dintre funcția f și modelul său pătratic, exprimată prin raportul

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{f(x_k) - q_k(s_k)}.$$

Dacă abaterea este mică $\rho_k \simeq 1$ și atunci Δ_k este crescut, în caz contrar Δ_k este scăzut.

Pachetele NAG și OPTIMA folosesc metoda căutării liniare, în timp ce pachetele IMSL și LANCELOT folosesc metoda regiunii de încredere.

Metoda Newton și celelalte două variante prezentate presupun pentru determinarea direcțiilor d_k rezolvarea la fiecare iterație a unui sistem liniar de n ecuații cu n necunoscute. Dacă se utilizează o metodă directă efortul de calcul are ordinul $O(n^3)$, pentru un număr mare de necunoscute. Din acest motiv se preferă utilizarea metodelor iterative de rezolvare, ca de exemplu, gradienti conjugați cu preconditionare bazată pe factorizarea Choleski incompletă. Tehnica astfel obținută este cunoscută sub numele de **metoda Newton trunchiată** deoarece iterațiile sunt întrerupte (trunchiate) atunci cnd este îndeplinit criteriul de oprire. Această tehnică este implementată în pachetele TN, VE08 și LANCELOT. În multe situații nu este disponibilă o rutină care să calculeze valorile exacte ale elementelor matricei Hessian. În astfel de situații există mai multe soluții, bazate pe o evaluare aproximativă a matricei Hessian, de exemplu prin diferențe finite. O altă abordare este cea specifică **metodelor de tip quasi-Newton**, la care la fiecare iterație se rezolvă sistemul liniar $B_{k+1}s_k = y_k$, în care B_{k+1} aproximează $\nabla^2 f(x_k)$. Cea mai des utilizată variantă de metodă de tip quasi-Newton este **metoda Broyden**, în care B_{k+1} se calculează în funcție de B_k , folosind relația

$$B_{k+1} = B_k - \frac{B_k s_k (B_k s_k)^T}{s_k^T B_k s_k} + \frac{y_k J_k^T}{y_k^T s_k} + \varphi_k [s_k^T B_k s_k] v_k v_k^T,$$

cu $\varphi_k \in (0, 1)$ și

$$v_k = \frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}.$$

Dacă se consideră $\varphi_k = 1$, rezultă **metoda DFP**, iar dacă se consideră $\varphi = 0$ rezultă **metoda BFGS**, care este considerată în prezent una din cele mai eficiente metode de optimizare.

Metodele de tip Broyden au avantajul că matricea B_k rămne pozitiv definită, att timp ct $y_k^T s_k > 0$, condiție ușor de verificat. Dintre pachetele care implementează această metodă menționăm NAG, IMSL, MATLAB, OPTIMA. Dacă se notează cu $M_k = B_k^{-1}$, rezultă următoarea corecție la iterația k :

$$d_k = -M_k \nabla f(x_k).$$

Metoda BFGS are o variantă în care se actualizează nu matricea B_k , ci inversa sa:

$$M_k = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) M_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + s_k s_k^T y_k^T s_k.$$

Răspunderea tot mai largă a metodelor de tip quasi-Newton a făcut ca metodele de tip "cea mai rapidă coborâre" să fie tot mai puțin folosite. Ambele tehnici cer cunoașterea doar a gradientului nu și a derivatelor de ordinul doi. Chiar dacă metodele de tip quasi-Newton cer mai multă memorie și mai mult timp de calcul pe o iterație, efortul este răsplătit de o convergență mult mai rapidă. Aparent, metoda BFGS nu este aplicabilă la probleme de mari dimensiuni, deoarece matricea B_k necesită n^2 locații de memorie. Dacă în schimb se alege matricea M_0 diagonală, s-ar putea reconstitui M_k folosind vectorii s_k și y_k de la iterațiile anterioare, cu un necesar de memorie doar de $2kn$ celule. O tehnică dedicată rezolvării problemelor de mari dimensiuni numită **quasi-Newton cu memorie limitată**, memorează vectorii s_k, y_k doar pe un număr limitat de iterații anterioare (de regulă cinci). Această tehnică este implementată în rutina LBFGS.

O altă clasă de algoritmi aplicabili în rezolvarea problemelor de optimizare de mari dimensiuni este cea a **a gradientilor conjugați**. La fiecare iterație se calculează $x_{k+1} = x_k + \alpha_k d_k$, în care α_k este determinat prin minimizarea de-a lungul direcțiilor d_k , care sunt calculate recursiv prin relațiile

$$d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k.$$

În varianta **Fletcher-Reeves**

$$\beta_k = \|\nabla f(x_{k+1})\|_2^2 / \|\nabla f(x_k)\|_2^2,$$

iar în varianta **Polak-Ribiere** (considerată mai performantă)

$$\beta_k = [\nabla f(x_{k+1}) - \nabla f(x_k)]^T \nabla f(x_{k+1}) / \|\nabla f(x_k)\|_2^2.$$

O îmbunătățire a performanțelor poate fi obținută prin "restartare", respectiv prin adoptarea periodică a valorii $\beta_k = 0$. Metoda gradientilor conjugați este implementată în pachetele NAG, IMSL și OPTIMA.

D.4 Optimizare neliniară cu restricții

Problema optimizării cu restricții de tip frontieră :

$$\min \{f(x) : l \leq x \leq u\}$$

reprezintă o clasă importantă de aplicații practice, la care fiecare parametru este inclus într-un interval închis de o limită minimă și de una maximă. Din acest motiv, rezolvării acestei probleme îi sunt dedicate mulți algoritmi și pachete software. Pentru rezolvarea acestei probleme au fost modificați algoritmi clasici de tip Newton, Newton-trunchiat,

quasi-Newton, dar și tehnicile de căutare liniară sau a regiunii de încredere folosite pentru extinderea domeniului de convergență al acestor algoritmi.

În principiu, rezolvarea problemelor cu restricții de tip frontieră se face prin metoda “**proiecției gradientului**”, în care $x_{k+1} = P[x_k - \alpha_k \nabla f(x_k)]$. Operatorul de proiecție P are componentele $P_i(x) = \text{med}(x_i, l_i, u_i)$, în care $\text{med}(x_i, l_i, u_i)$ este valoarea mediană a mulțimii formate din cele trei elemente. În acest fel căutarea pe direcția gradientului se face doar în interiorul domeniului de definiție al funcției obiectiv. În acest caz regiunea de încredere se obține prin intersecția elipsoidului definit de parametrul Δ_k și domeniul paralelipipedic de definiție a funcției obiectiv, urmnd ca subproblema de optimizare pătratică ce trebuie rezolvată la fiecare iterație să aibă forma:

$$\min \{q_k(s) : \|D_k s\|_2 \leq \Delta_k, l \leq x_k + s \leq u\}.$$

Implementări ale acestor algoritmi se găsesc în NAG, IMSL, MATLAB, LANCELOT, OPTIMA, TN și VE08. Problema generală de **optimizare neliniară cu restricții**:

$$\min \{f(x) : c_i(x) \leq 0, i \in I, c_i(x) = 0, i \in E\}$$

se întâlnește de obicei într-o formă mai simplă în care **restricțiile de tip inegalitate sunt de tip frontieră**:

$$\min \{f(x) : c(x) = 0, l \leq x \leq u\}$$

și în care $c : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Principalele tehnici utilizate pentru problemele cu restricții sunt: metodele de tip gradient redus, programarea liniară secvențială, programare pătratică secvențială și metoda bazată pe penalități sau pe **Lagrangean augmentat**. Un rol fundamental în înțelegerea acestor tehnici îl joacă **funcția Lagrangean** definită ca:

$$L(x, \lambda) = f(x) + \sum_{i \in I \cup E} \lambda_i c_i(x).$$

Aceasta sumează att funcția obiectiv f ct și restricțiile c_i , de tip egalitate sau inegalitate, fiecare ponderată cu un număr real λ_i numit multiplicator. Condiția necesară de existență a unui minim local x^* pentru problema cu restricții, constă în existența unor valori λ_i^* pentru **multiplicatorii Lagrange**, astfel înct să fie satisfăcute condiția de prim ordin:

$$\nabla_x L(x^*, \lambda^*) = \nabla f(x^*) + \sum_{i \in A^*} \lambda_i^* \nabla c_i(x^*) = 0,$$

în care $A^* = \{i \in I : c_i(x^*) = 0\} \cup E$ se numește mulțimea activă, și în plus trebuie ca $\lambda_i^* \geq 0$ dacă $i \in A^* \cap I$. Condiția suficientă de existență a minimului se referă la Hessianul lagrangeanului:

$$\nabla_{xx}^2 L(x^*, \lambda^*) = \nabla^2 f(x^*) + \sum_{i \in A^*} \lambda_i^* \nabla^2 c_i(x^*),$$

care trebuie să fie pozitiv definit, respectiv trebuie ca:

$$w^T \nabla_{xx}^2 L(x^*, \lambda^*) w > 0,$$

pentru orice vector w nenul din mulțimea:

$$\{w \in \mathbb{R}^n : \nabla c_i(x^*)^T w = 0, i \in I_+^* \cup E; \nabla c_i(x^*)^T w \leq 0, i \in I_0^*\}$$

în care $I_+^* = i \in A^* \cap I : \lambda_i^* > 0, I_0^* = i \in A^* \cap I : \lambda_i^* = 0$.

Metoda programării pătratică secvențiale (SQP) se bazează pe metoda Newton, minimiznd la fiecare iterație modelul pătratic:

$$q_k(d) = \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla_{xx}^2 L(x_k, \lambda_k) d,$$

și înlocuind de obicei restricțiile cu aproximațiile lor liniare:

$$\min \{q_k(d) : c_i(x_k) + \nabla c_i(x_k)^T d \leq 0, i \in I, c_i(x_k) + \nabla c_i(x_k)^T d = 0, i \in E\},$$

urmnd ca soluția d_k să reprezinte pasul efectuat $x_{k+1} = x_k + d_k$.

Varianta prezentată poartă numele de **programare secvențială pătratică bazată pe inegalități (IPQ)**. O altă variantă, **bazată pe egalități (EPQ)**, rezolvă la fiecare iterație problema de programare pătratică:

$$\min \{q_k(d) : c_i(x_k) + \nabla c_i(x_k)^T d = 0, i \in W_k\}$$

și determină pasul d_k . Această variantă menține explicit o mulțime de lucru W_k , formată din indicii activi la iterația curentă (indică restricții de tip inegalitate care sunt active în timpul rezolvării problemei de programare pătratică). Mulțimea W_k este actualizată la fiecare iterație, examinnd valorile lui $c_i(x_{k+1})$ pentru i ce nu aparține lui W_k la iterația $k + 1$ precum și valorile multiplicatorilor Lagrange. Mai multe pachete ca de exemplu OPTIMA, MATLAB și SQP sunt bazate pe acest algoritm. Convergența algoritmilor QP secvențiali poate fi îmbunătățită prin folosirea căutării liniare. Spre deosebire de cazul optimizărilor fără restricție la care mărimea pasului era aleasă ca o aproximație a poziției pe direcția de căutare a minimului funcției f , acum trebuie verificat și felul în care acest punct satisface restricțiile. Deoarece aceste două ținte sunt deseori în conflict, este necesară definirea unui criteriu care să indice dacă un punct este mai bun dect altul sau nu. Acest lucru este realizat prin **funcția de penalitate (sau de merit)**, care ponderează importanța relativă a diferitelor ținte, ca de exemplu cea folosită în MATLAB și SQP:

$$P_1(x, \nu) = f(x) + \sum_{i \in E} \nu_i |c_i(x)| + \sum_{i \in I} \nu_i \max(c_i(x), 0),$$

în care $\nu_i > 0$ sunt parametri de penalizare, sau funcția Lagrangean augmentată, folosită în NLPQL și OPTIMA:

$$L_A(x, \lambda, \nu) = f(x) + \sum_{i \in E} \lambda_i c_i(x) + \frac{1}{2} \sum_{i \in E} \nu_i c_i^2(x) + \sum_{i \in E} \psi_i(x, \lambda, \nu),$$

în care

$$\psi_i(x, \lambda, \nu) = \frac{1}{\nu_i} \{ \max \{0, \lambda_i + \nu_i c_i(x)\}^2 - \lambda_i^2 \}$$

Unii algoritmi, în loc să utilizeze funcția de penalizare pentru a verifica acceptabilitatea pasului propus de algoritmul QP, minimizează direct valoarea acestei funcții. Este cazul **algoritmului lagrangeanului augmentat**, care minimizează succesiv funcția L_A , față de x și actualizează valorile λ și eventual ν . În cazul în care restricțiile inegalitate sunt de tip frontieră se preferă augmentarea lagrangeanului doar cu restricția egalitate:

$$L_A(x, \lambda, \nu) = f(x) + \sum_{i \in E} \lambda_i c_i^2(x),$$

urmînd ca x_{k+1} să se determine prin rezolvarea subproblemei de optimizare cu restricții de tip inegalitate:

$$\min \{L_A(x, \lambda_k, \nu_k) : l \leq x \leq u\},$$

iar multiplicatorii λ_i să fie actualizați cu expresia $\lambda_i + \nu_i c_i(x_k)$. Această metodă este implementată în LANCELOT, OPTIMA și OPTPACK. Folosirea parametrilor de penalizare poate fi evitată prin căutarea doar de-a lungul curbelor care se află în sau foarte aproape de mulțimea de fezabilitate (în care sunt îndeplinite restricțiile).

Această abordare este specifică **metodei gradientului redus**, care utilizează restricțiile de tip egalitate pentru a elimina o parte din variabile, de exemplu primele (într-un număr egal cu dimensiunea mulțimii E), variabile numite "de bază" și notate x_B . Dacă se notează cu x_N vectorul variabilelor care nu sunt de bază, atunci $x_B = h(x_N)$ cu aplicația h definită implicit de restricția de tip egalitate prin $c[h(x_N), x_N] = 0$. În realitate aplicația nu este determinată explicit, ci prin actualizări de tip Newton, realizată cu atribuirea:

$$x_B \leftarrow \partial_{BC}(x_B, x_N)c(x_B, x_N)$$

în care ∂_{BC} este matricea Jacobian a funcțiilor c față de variabilele de bază. Problema inițială de optimizare neliniară cu restricții este acum transformată într-o problemă fără restricții egalitate și cu restricții de tip frontieră:

$$\min \{f(h(x_N), x_N) : l_N \leq x_N \leq u_N\}.$$

Variabilele x_N sunt împărțite în continuare în două categorii: variabile fixe x_F , care vor fi menținute constante la iterația curentă, deoarece ele sunt, fie la limita inferioară, fie la cea superioară și variabile x_S , care sunt libere să se modifice la iterația curentă.

Algoritmul gradientilor reduși, în forma sa standard este aplicat în CONOPT, unde căutarea se face la fiecare iterație în subspațiul variabilelor superbazice, după direcția celei mai rapide coborri. MINOS și GRG folosesc generalizări ale metodei gradientilor reduși, bazate pe aproximări BFGS dense ale matricei Hessian a lui f în raport cu variabilele superbazice x_S în timp ce LSGRG folosește varianta de memorie redusă. Tehnica folosită în MINOS este cunoscută și sub numele de **metoda proiecției lagrangeanului augmentat**, deoarece nu aplică direct metoda gradientilor proiectați în rezolvarea problemei generale de optimizare ci doar în subproblema rezolvată la fiecare iterație și care este cu restricții liniare.

În metoda gradientului redus, o mare parte din efort se referă la inversarea Jacobianului $\partial_{BC}(x_B, x_N)$. Aceasta este de obicei substituită cu factorizarea LU. În GRGZ se realizează o factorizare densă de matrice plină, în timp ce în CONOPT, MINOS și LSGRGZ sunt utilizate tehnici de factorizare pentru matrici rare, care permit rezolvarea unor probleme de mari dimensiuni. Cnd o parte din restricții sunt liniare, atunci problema se simplifică deoarece termenii corespunzători din matricea Jacobian rămân constanți pe parcursul iterațiilor. Numeroase coduri ca MINOS și unele subrutine NAG sunt capabile să exploateze avantajele ce rezultă din prezența unor restricții liniare, în timp ce bibliotecile matematice IMSL și PORT3 conțin subrutine speciale dedicate optimizării neliniare cu restricții liniare. Să menționăm în final **algoritmul programării pătraticе secvențiale fezabile** implementat în FSPQ. În acest algoritm pasul la fiecare iterație este definit ca o combinație dintre direcția QP secvențială, o direcție strict fezabilă (care indică interiorul mulțimii de fezabilitate, în care toate restricțiile sunt îndeplinite), și eventual o corecție de ordinul doi. Această combinație este aleasă astfel încât rezultatul să se afle în mulțimea de fezabilitate păstrându-se pe ct posibil proprietățile de convergență locală rapidă. Execuția acestor algoritmi este mult mai scumpă decât a algoritmului clasic SQP, în schimb, datorită faptului că funcția de merit este chiar f , el este de preferat atunci cnd este imposibil de calculat f în afara zonei de fezabilitate sau cnd oprirea în afara zonei de fezabilitate (lucru ce se poate întâmpla cu ceilalți algoritmi) este inacceptabilă.

D.5 Problema celor mai mici pătrate neliniare

Bibliotecile matematice conțin în secțiunea de optimizare nu numai algoritmi pentru probleme generale dar și o bogată serie de algoritmi dedicați unor clase mai restrnse de probleme. Una dintre acestea este **problema celor mai mici pătrate neliniare** de forma:

$$\min \{r(x) : x \in \mathbb{R}^n\}$$

cu $r(x) = \frac{1}{2} \|f(x)\|_2^2$, în care $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Această problemă apare deseori în aproximarea funcțională sau interpolarea datelor. În această problemă gradientul și matricea

Hessian ale funcției scalare r au formulele:

$$\nabla r(x) = f'(x)^T f(x),$$

$$\nabla^2 r(x) = f'(x)^T f'(x) + \sum_{i=1}^m f_i(x) \nabla^2 f_i(x),$$

în care

$$f'(x) = (\partial_1 f(x), \dots, \partial_n f(x))$$

este matricea Jacobian a aplicației f . În multe cazuri primul termen din $\nabla^2 r$ este mai important decât al doilea, mai ales atunci când valorile $f_i(x)$ sunt mici. Algoritmul care exploatează această observație este cunoscut sub numele **Gauss-Newton**. Pentru a obține o nouă iterație $x_{k+1} = x_k + \alpha_k d_k$, se efectuează o căutare liniară după direcțiile d_k , pentru a determina scalarul d_k corespunzător minimului funcției $r(x)$. Direcția de căutare d_k se determină prin rezolvarea sistemului liniar:

$$(f'(x_k)^T f'(x_k)) d_k = -f'(x_k)^T f(x_k)$$

Dacă $f'(x_k)$ are rangul mai mic decât n acest sistem poate avea mai multe soluții, dintre care se poate alege de exemplu:

$$d_k = -f'(x)^+ f(x_k)$$

în care semnul $+$ indică pseudoinversa matricei (corespunzătoare soluției cu norma l_2 minimă). În pachete ca MATLAB, NAG sau OPTIMA sunt implementate tehnici mai puțin costisitoare, de exemplu cea de perturbare a matricei sistemului

$$(f'(x_k)^T f'(x_k) + s_k) d_k = -f'(x_k)^T f(x_k),$$

cu matricea s_k aleasă astfel încât să se obțină convergența globală, în timp ce este garantată convergența locală rapidă (de exemplu o aproximare a celui de-al doilea termen din $\nabla^2 r$).

Algoritmul Levenberg-Marquardt este unul dintre cei mai puternici pentru rezolvarea problemelor celor mai mici pătrate neliniare. El reprezintă o modificare a algoritmului Gauss-Newton bazată pe tehnica regiunii de încredere. Soluția subproblemei

$$\min \left\{ \frac{1}{2} \|f'(x_k)s + f(x_k)\|_2^2 : \|D_k s\|_2 \leq \Delta_k \right\}$$

aflată în regiunea de încredere este folosită la determinarea noii iterații $x_{k+1} = x_k + s_k$. Ea satisface sistemul liniar:

$$(f'(x_k)^T f'(x_k) + \lambda_k D_k^T D_k) s_k = -f'(x_k)^T f(x_k)$$

pentru unele valori $\lambda_k \geq 0$. Multiplicatorul Lagrange λ_k este nul dacă pasul are norma mai mică decât Δ_k , altfel D_k este ales astfel încât $\|D_k s_k\|_2 = \Delta_k$. Sistemul liniar reprezintă forma normală a următoarei probleme de cele mai mici pătrate:

$$\min \left\{ \left\| \begin{bmatrix} f'(x_k) \\ \lambda_k^{\frac{1}{2}} D_k \end{bmatrix} + \begin{bmatrix} f(x_k) \\ 0 \end{bmatrix} \right\|_2^2 : s \in \mathbb{R}^n \right\},$$

pentru care se pot aplica în mod eficient factorizări bazate pe transformări Householder sau Givens. Atunci algoritmul Gauss-Newton și Levenberg-Marquardt prezintă convergență pătratică pentru probleme cu reziduul nul $r(x^*) = 0$, și convergența liniară în caz contrar (când este preferabil algoritmul BFGS, care are convergența supraliniară).

Algoritmul Levenberg-Marquardt este implementat în MINPACQ, IMSL și MATLAB. Unele implementări aplică strategii hibride care combină metode de tip Gauss-Newton (aplicată de exemplu doar dacă reziduul scade de cel puțin 5 ori la fiecare iterație) cu metode de tip quasi-Newton.

D.6 Rezolvarea sistemelor de ecuații neliniare

O altă clasă particulară de probleme care beneficiază de tehnica programării neliniare, o reprezintă **sistemele de ecuații neliniare**:

$$f(x) = 0$$

în care $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Acestea apar fie ca restricție în problemele de optimizare, fie în urma discretizării ecuațiilor diferențiale sau integrale, cum se întâmplă în cazul problemelor de cmp. Rezolvarea lor poate fi privită ca o problemă de cele mai mici pătrate neliniare deoarece soluția sistemului coincide cu soluția globală a problemei de minimizare:

$$\min \{ \|f(x)\| : x \in \mathbb{R}^n \},$$

în care $\|\cdot\|$ este de obicei norma l_2 din \mathbb{R}^n .

Metoda Newton aplicată în rezolvarea ecuațiilor neliniare are drept corecție la iterația k soluția sistemului liniar:

$$f'(s_k)s_k = -f(x_k),$$

și $x_{k+1} = x_k + s_k$. Convergența locală rapidă (pătratică) reprezintă marele avantaj al metodei Newton. Dezavantajele metodei Newton includ necesitatea calculului matricei Jacobian $f'(x_k)$ de dimensiuni $n \times n$ la fiecare iterație și lipsa convergenței globale (atunci când inițializarea este departe de soluție).

Pachetele de programe încearcă să elimine cele două dezavantaje ale metodei, utiliznd în locul matricei Jacobian aproximări ale acesteia și folosind cele două strategii de bază pentru extinderea domeniului de convergență: **căutarea liniară** (GAUSS, MATLAB, OPTIMA) și **regiunea de încredere** (IMSL, NAG, LANCELOT și MINPACK). În cazul problemelor de mari dimensiuni unele pachete cum este NITSOL folosesc metoda Newton trunchiată, rezolvnd iterativ sistemul liniar cu o precizie relativă a reziduului raportată în normă la termenul liber scăznd pe măsură ce x_k se apropie de soluția x^* . O metodă eficientă de reducere a efortului de calcul constă în folosirea **metodelor quasi-Newton** bazate pe formula

$$B_{k+1}(x_{k+1} - x_k) = f(x_{k+1}) - f(x_k)$$

de tip Broyden, în care matricea B_k se actualizează recurent cu formula

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T}{\|s_k\|_2^2},$$

unde $s_k = x_{k+1} - x_k$ și $y_k = f(x_{k+1}) - f(x_k)$. Inițializarea B_0 este de obicei o aproximare cu diferențe finite a matricei Jacobian. Aspectul remarcabil al metodei Broyden constă în faptul că ea este capabilă să genereze aproximări rezonabile ale matricei Jacobian, fără evaluări suplimentare ale funcției f . Pachetele NAG, IMSL și MINPACK folosesc metoda Broyden combinată cu utilizarea tehnicii regiunii de încredere.

Algoritmii care utilizează strategiile de căutare liniară a regiunii de încredere urmăresc satisfacerea inegalității $\|f(x_{k+1})\| \leq \|f(x_k)\|$ la fiecare iterație k . Din acest motiv iterațiile pot fi atrase într-un minim local z^* pentru care $f(z^*) \neq 0$ și să nu găsească soluția x^* a sistemului de ecuații $f(x) = 0$. Pentru a garanta convergența către x^* indiferent de inițializare sunt necesare metode mai complexe cum este **metoda continuității (homotopiei)**. Aceste metode necesită un efort de calcul mai mare, dar sunt utile atunci cnd este dificil de găsit o inițializare corespunzătoare. Ideea metodei este de a porni de la o problemă ușoară (de exemplu, una liniară), care este transformată gradual către problema neliniară dificilă $f(x) = 0$. De exemplu, dacă se introduce un parametru scalar λ și se definește funcția

$$h(x, \lambda) = f(x) - (1 - \lambda)f(x_0),$$

în care x_0 este un punct dat din \mathbb{R}^n , atunci problema este rezolvată succesiv pentru diferite valori ale lui λ începnd de la 0 și terminnd cu 1. Cnd $\lambda = 0$, soluția este clar x_0 , iar cnd $\lambda = 1$, soluția este cea a problemei originale $f(x) = 0$. Avantajul constă în faptul că atunci cnd se rezolvă ecuația, pentru o valoare a parametrului λ , se consideră ca inițializare soluția obținută pentru valoarea precedentă a parametrului λ . În cazul problemelor de cmp se poate considera λ ca fiind procentul din sursa de cmp, astfel înct domeniul este excitat progresiv de la cmp nul către cmpul maxim. Pachetele HOMPACK și PITCON implementează această metodă determinând noua iterație (x_{k+1}, λ_{k+1})

pornind de la iterația curentă (x_k, λ_k) și rezolvnd sistemul augmentat de forma

$$h(x, \lambda) = 0, w_k^T x + \mu_k \lambda - t_k = 0,$$

în care ecuația liniară suplimentară este aleasă astfel încât (w_k, μ_k) este vector unitar în \mathbb{R}^{n+1} iar t_k este valoarea țintă a uneia dintre componente (x_{k+1}, λ_{k+1}) care a fost fixată de un pas predictor.

D.7 Programare liniară

Categoria de probleme de optimizare, aparent cea mai simplă, dar în același timp cea mai des întâlnită în practică, este cea a **programării liniare**(LP). În acest caz att funcția obiectiv ct și restricțiile au un caracter liniar:

$$\min \{c^T x : b_l \leq Ax \leq b_n, l \leq x \leq n\},$$

în care A este o matrice de dimensiune $m \times n$, iar c și x sunt vectori de n -dimensionali.

Problema **optimizării rețelelor** de distribuție, transport sau comunicație reprezintă cea mai importantă sursă de aplicații practice pentru problmele LP. O rețea liniară este definită de un graf, alcătuit dintr-o mulțime N de noduri ($i = 1, 2, \dots, n$) și o mulțime A de laturi. Necunoscutele problemei reprezintă debitele x_{ij} asociate laturii (i, j) , ce unește nodul i cu nodul j . Presupunnd că fiecare latură are un cost unitar C_{ij} (pentru distribuție, transport sau comunicație), problema minimizării costului traficului într-o relație constă în determinarea debitelor x_{ij} astfel încât costul total să fie minim:

$$\min \sum_{(i,j) \in A} C_{ij} x_{ij},$$

în condițiile în care sunt respectate restricțiile:

$$l_{ij} \leq x_{ij} \leq u_{ij}, \quad (i, j) \in A;$$

$$\sum_j x_{ij} - \sum_j x_{ji} = s_i, \quad 1 \leq i \leq n.$$

Restricția de tip egalitate se referă la debitul net s_i al fiecărui nod, pe când cea de tip inegalitate se referă la capacitatea laturilor rețelei. Problema **debitului maxim** într-o rețea constă în maximizarea debitului total între două noduri, unul sursă și altul destinație, în condițiile respectării capacității laturilor. În problema **atribuirii**, nodurile sunt împărțite în N_1 noduri persoane și N_2 noduri obiecte, căutndu-se o corespondență biunivocă între N_1 și N_2 , care să asigure suma totală minimă a costurilor C_{ij} ale laturilor

ce definesc această corespondență. Prin schimbări de variabile problema programării liniare poate fi adusă la **forma standard LP**:

$$\min \{c^T x : Ax = b, x \geq 0\},$$

în care $x \in \mathbb{R}^n$ este vectorul necunoscutelor, $c \in \mathbb{R}^n$ este vectorul de cost și $A \in \mathbb{R}^{m \times n}$ este matricea restricțiilor liniare.

Regiunea de fezabilitate descrisă de restricții este un poliedru convex numit și simplex. Soluția problemei se află pe frontiera acestui simplex, de obicei într-un vrf al lui.

Metoda clasică de rezolvare a problemei LP este **metoda simplex**, descoperită de Dantzig în 1940. Ea se bazează pe căutarea în mulțimea vrfurilor simplexului. Pornind de la o inițializare arbitrară la punctul de căutare într-un vrf adiacent cu o valoare a funcției de cost (obiectiv) $f(x) = c^T x$ mai mică, eventual de-a lungul laturii care determină scăderea cea mai rapidă a funcției de cost. Cnd toate vrfurile vecine au valorile mai mari decât ale funcției de cost, căutarea este încheiată.

Transpunerea algebrică și algoritmică a acestei metode se bazează pe observația că cel puțin $(m-n)$ componente ale lui x sunt nule, dacă x este vrful mulțimii de fezabilitate. În consecință, în fiecare vrf componentele lui x pot fi împărțite în n variabile de bază (toate nenegative) și $(m-n)$ variabile nule. Fie $x = [x_B, x_N]$ cu componente $x_B \in \mathbb{R}^m, x_N \in \mathbb{R}^{m-n}$, care poziționează coloanele matricei A în $[B \times N]$, în care B este matrice pătrată. Rezultă că

$$x_B = B^{-1}(b - Nx_N).$$

La fiecare iterație a metodei simplex, x_B și x_N schimbă între ele o componentă, ceea ce corespunde mutării într-un vrf adiacent. Partiționarea vectorului de cost c în $[c_B, c_N]$ conduce la următoarea expresie a funcției obiectiv:

$$c^T x = c_B^T x_B + c_N^T x_N = c_B^T B^{-1}b + [c_N - N^T B^T c_B]^T x_N,$$

în care $d_N = c_N - N^T B^T c_B$ este numit vectorul de cost redus. Deoarece acea componentă a lui x_N care va fi transferată în x_B va putea deveni pozitivă, pentru scăderea funcției de cost $c^T x$ va trebui aleasă pentru schimbare acea componentă i care corespunde unei componente d_N negative. Cea mai simplă strategie este de a alege pe i corespunzător celei mai negative componente din vectorul d_N . Dacă toate componentele vectorului d_k sunt pozitive, atunci punctul optimal a fost găsit.

O strategie mai rafinată ar fi cea a **laturii cu cea mai rapidă coborere**, dar care necesită un efort mai mare la fiecare iterație.

Deoarece elementele lui x_N vor fi nule cu excepția lui x_{Ni} , rezultă $x_B = B^{-1}(b - N_i x_{Ni}) \geq 0$, de unde rezultă în mod explicit:

$$x_{N_i} = \min \left\{ \frac{(B^{-1}b)_j}{(B^{-1}N_i)_j} : (B^{-1}N_i)_j > 0 \right\},$$

dacă x_{N_i} este cea mai mare valoare care menține inegalitatea $x_B \geq 0$. Indexul j ce conduce la minimumul din formula anterioară indică variabila bazică x_{B_j} care va deveni nebazică la ultimul pas. Dacă mai multe componente ating minimumul simultan, atunci este selectată cea pentru care $(B^{-1}N_i)_j$ are valoare maximă.

Majoritatea calculelor în algoritmul simplex se referă la evaluarea produselor $B^{-1}C_B$ și $B^{-1}N_i$ precum și la necesitatea menținerii urmei schimbărilor bazei asupra matricelor B și B^{-1} . O tehnică folosită în multe coduri comerciale pentru a scădea efortul de calcul constă în utilizarea factorizării $B = PLUQ$, în care P și Q sunt matrici de permutare iar L și U sunt matrici triunghiulare inferior respectiv superior, menținerea unei modificări făcându-se prin matrici de permutare. Deoarece $P^T P = P P^T = I$ și $Q^T Q = Q Q^T = I$, produsul $z = B^T C_B$ se calculează prin următoarea secvență de operații:

$$U^T z_1 = Q C_B, \quad L^T z_2 = z_1, \quad z = P z_2.$$

Cnd matricei B i se modifică o singură coloană factorizarea poate fi actualizată printr-un număr redus de operații elementare, care pot fi memorate compact. Cnd memoria necesară acestor operații devine excesivă se face factorizarea.

Dintre pachetele de programe care conțin implementări ale metodei simplex menționăm: IMSL, NAG, CPLEX, FortLP, LINDO, MINOS și OSL.

Marele dezavantaj al metodei simplex constă în complexitatea sa exponențială (numărul de iterații poate crește în cazurile cele mai defavorabile factorial cu numărul de variabile), ceea ce o face nepotrivită pentru probleme de foarte mari dimensiuni.

Din acest motiv, încă din anii 1970 au început cercetări pentru găsirea de algoritmi cu ordin de complexitate polinomial pentru rezolvarea problemei LP. Aceștia se bazează nu pe parcurgerea nodurilor ci pe căutarea iterativă folosind puncte interioare din domeniul de fezabilitate.

Dintre **metodele de punct iterativ** cea mai promițătoare pare a fi metoda bazată pe punctele interne **primare-duale**. În această metodă se rezolvă pe lngă forma standard a problemei "primare" și problema duală:

$$\max \{b^T y : A^T y \leq 0\},$$

urmărind să se găsească tripletul (x, y, s) , pentru care:

$$Ax = b, \quad A^T y + s = 0, \quad S X e = 0$$

cu $x \geq 0, s \geq 0$, și $S = \text{diag}(s_1, s_2, \dots, s_n)$, $X = \text{diag}(x_1, x_2, \dots, x_n)$ iar e este un vector cu toate componentele unitate. Metoda primar-dual poate fi concepută ca o variantă a metodei Newton aplicată rezolvării acestui sistem de trei ecuații.

Se generează iterațiile (x_k, y_k, s_k) cu $x_k > 0, s_k > 0$, puncte interioare în domeniul de fezabilitate, astfel încât pe măsură ce $k \rightarrow \infty$, valoarea restricțiilor $\|Ax_k - b\|$ și $\|A^T y_k + s_k - c\|$ să tindă spre zero, împreună cu $x_k^T s_k$ numită ”distanță de dualitate”. Ordinul de complexitate al acestui algoritm este mai mic de $O(n^{\frac{3}{2}})$. Implementări ale metodelor de punct interior se găsesc în pachetele OB1, OSL și KORBX.

D.8 Programarea pătratică

Problema **programării pătratice** (QP) constă în minimizarea unui polinom de gradul doi de n variabile supus unor restricții liniare de tip inegalitate sau egalitate:

$$\min \left\{ \frac{1}{2} x^T Q x + c^T x : a_i^T x \leq b_i, i \in I, a_i^T x = b_i, i \in E \right\},$$

în care $Q \in \mathbb{R}^{n \times n}$ este o matrice simetrică iar I și E sunt submulțimi de indici.

Problema **celor mai mici pătrate liniare** este un caz particular al programării pătratice care se întâlnește frecvent în aplicațiile de prelucrarea datelor (aproximarea funcțiilor). În acest caz se caută:

$$\min \left\{ \frac{1}{2} \|C_k - d\|_2^2 : a_i^T x \leq b_i, i \in I, a_i^T x = b_i, i \in E \right\}$$

în care $C \in \mathbb{R}^{m \times n}$ și $d \in \mathbb{R}^n$. Chiar dacă cele două probleme sunt similare ($Q = C^T C$ și $c = C^T d$), ultima are metode specifice care evită calculul explicit al matricei $Q = C^T C$, mai prost condiționată decât C (de exemplu în pachetul LSSOL se folosește descompunerea QR a matricei C).

Dificultatea rezolvării problemei QP depinde de natura matricei Q .

Problemele de **programare pătratică convexă** sunt ușor de rezolvat, deoarece Q este pozitiv semidefinită și este mărginită inferior. În acest caz, funcția are un singur minim local iar în absența restricțiilor acesta se obține prin rezolvarea sistemului liniar de ecuații $Qx = -c$.

Dacă Q are și valori proprii nenegative, atunci **programarea pătratică este neconvexă** și funcția obiectiv are minimele pe frontiera domeniului. O metodă des folosită este **metoda mulțimilor active**, care se bazează pe căutarea soluției de-a lungul laturilor sau fețelor mulțimii de fezabilitate, prin rezolvarea unei secvențe de probleme QP cu restricții de tip egalitate:

$$\min \left\{ \frac{1}{2} x^T Q x + c^T x : Ax = b \right\}.$$

Metoda **spațiului nul** pentru rezolvarea acestei probleme constă în găsirea, prin factorizare ortogonală, a unei matrice $Z \in \mathbb{R}^{n \times m}$ de rang maximal, care generează spațiul

nul al matricei A . În aceste condiții, pornind de la un vector fezabil x_0 , se poate construi alt vector fezabil $x = x_0 + Zw$, în care $w \in \mathbb{R}^m$.

Problema QP cu restricție de tip egalitate poate fi pusă deci sub forma unei probleme QP fără restricție:

$$\min \left\{ \frac{1}{2} w^T (Z^T Q Z) w + (Qx_0 + c)^T Z w : w \in \mathbb{R}^m \right\}.$$

Dacă matricea Hessian $Z^T Q Z$ este pozitiv definită, atunci unica soluție w a acestei probleme se obține prin rezolvarea sistemului liniar:

$$(Z^T Q Z) w = -Z^T (Qx_0 + c).$$

După rezolvarea acestui sistem se obține $x = x_0 + Zw$. Revenind la metoda mulțimilor active, la fiecare iterație se pornește de la un punct x_k în raza de fezabilitate și se determină direcții d_k prin rezolvarea problemei

$$\min \{ q(x_k + d) : a_i^T (x_k + d) = b_i, i \in W_k \}$$

cu $q(x) = \frac{1}{2} x^T Q x + c^T x$, iar $W_k \subset A = E \cup \{i \in I : a_i^T x_k = b_i\}$ o mulțime (de indici) de lucru, parte dintr-o mulțime de indici activi A .

Mulțimea de lucru W_k este actualizată la fiecare iterație, urmnd ca în final ea să coincidă cu A^* , mulțimea restricțiilor active ale soluției problemei x^* . Folosind soluția d_k se calculează cel mai mare pas posibil care nu contravine nici unei restricții:

$$M_k = \max \{ (b_i - a_i^T x_k) / (a_i^T d_k) : a_i^T d_k > 0, i \notin W_k \}$$

și se determină $x_{k+1} = x_k + \alpha_k d_k$, cu $\alpha_k = \min \{1, \mu_k\}$. Noua mulțime de lucru W_{k+1} se obține introducnd toate restricțiile active pentru x_{k+1} . Dacă $d_k = 0$, atunci procesul iterativ se oprește. Acest algoritm este implementat în IMSL (QP convexă și densă), NAG(QPOPT), MATLAB(QP densă neconvexă) și LINDO(QP cu matrice rară), pentru probleme de mari dimensiuni.

D.9 Arborele de decizie pentru metoda deterministă

Din cele prezentate, rezultă următoarea clasificare a problemelor de optimizare (minimizare), care reprezintă în același timp și arborele de decizie pentru alegerea metodelor de rezolvare:

1. După numărul de variabile ale funcției obiectiv:

- unidimensională:

- multidimensională de mică mărime ($n < 150$);
- multidimensională de mărime medie ($150 \leq n < 1000$);
- multidimensională de scară mare ($n \geq 1000$).

2. După tipul variabilelor funcției obiectiv:

- continue (optimizare continuă);
- discrete (optimizare întreagă);
- mixte (optimizare hibridă).

3. După felul funcției obiectiv:

- liniară (programarea liniară LP);
- pătratică (programarea pătratică QP);
- neliniară (programarea neliniară NLP).

În ultimul caz funcția poate fi:

- unimodală (are un singur minim local, implicit global), eventual convexă;
- cu un singur minim global dar cu mai multe minime locale;
- cu mai multe minime globale și eventual cu unele locale.

Valoarea minimului funcției obiectiv poate fi cunoscută, de exemplu $\min(f) = 0$, dar nu și poziția sa, sau aceasta poate fi necunoscută, cum se întâmplă de obicei.

4. După netezimea funcției obiectiv:

- discontinuă;
- continuă, dar nu și derivabilă;
- cu derivata continuă, dar fără derivata a doua;
- cu derivata a doua continuă.

În ultimul caz se pot folosi metode de tip Newton bazate pe matricea Hessian, dacă este disponibilă o rutină de calcul a derivatei.

În primele două cazuri nu se pot utiliza metode bazate pe derivate. Primul caz este cel mai dificil, deoarece în general nu se poate garanta corectitudinea soluției numerice (un minim se poate afla la o distanță oricât de mică, inclusiv sub "zeroul mașinii" față de un maxim). În al doilea caz, dacă funcția este Lipschitziană, atunci are o viteză de creștere mărginită și cunoscând valoarea funcției obiectiv într-un punct se pot deduce valorile maxime și minime pe care le poate lua funcția într-o vecinătate sferică de rază dată.

5. După tipul restricțiilor:

- probleme fără restricții;
- probleme cu restricții de tip frontieră;
- probleme cu restricții de tip inegalitate;
- probleme cu restricții de tip egalitate;
- probleme cu toate tipuri de restricții.

În al doilea caz restricțiile sunt inegalități impuse variabilelor $l \leq x \leq u$, domeniul de căutare avnd formă paralelipipedică (eventual cu unele laturi la infinit).

În cazul restricțiilor de tip inegalitate $c(x) \leq 0$, iar în cazul celor de tip egalitate $c(x) = 0$. În ultimele trei categorii de restricții, acestea pot fi de două categorii:

- restricții liniare, funcția c , definitorie a restricției, este liniară;
- restricții neliniare, funcția c , definitorie a restricției, este neliniară.

Restricțiile de tip frontieră sunt cazuri particulare de restricții liniare de tip inegalitate. Se pot imagina probleme cu restricții liniare de tip egalitate dar avnd restricții neliniare de tip inegalitate, combinate eventual cu restricții de tip frontieră.

În problemele de programare liniară ca și în cele de programare pătratică funcția obiectiv este liniară, respectiv pătratică, în schimb restricțiile sunt întotdeauna liniare.

Dacă numărul restricțiilor de tip egalitate este mai mare decât numărul variabilelor este posibil ca problema să fie incompatibilă și deci să nu aibă soluție.

În problemele liniare (cu excepția cazului în care funcția obiectiv este construită) și în unele probleme neliniare, funcția obiectiv nu este mărginită inferior și în consecință, problema optimizării nu are soluție. Prezența restricțiilor de tip inegalitate determină existența soluției acestor probleme, soluție care va fi plasată pe frontiera domeniului de căutare.

Anexa E

Metode stocastice pentru optimizarea globală

E.1 Algoritmul genetic canonic

Algoritmii genetici(GA) reprezintă o familie de modele inspirate din evoluția speciilor biologice. Acești algoritmi urmăresc evoluția unei populații formată din indivizi care reprezintă soluții potențiale ale problemei de optimizare. Un individ este reprezentat bi-univoc printr-o structură de date numită cromozomul individului respectiv. În urma încrucișărilor, mutațiilor și selecției celor mai "buni" indivizi (cei cu cea mai bună valoare a funcției obiectiv), populația evoluează, apropiindu-se de soluția problemei. Multe din operațiile efectuate de un algoritm genetic au un caracter stocastic (ca de exemplu, răspunderea aleatoare a populației inițiale, mutația, încrucișarea probabilistică și chiar selecția), ceea ce face ca acești algoritmi să evite eșuarea întregii populații într-un minim local.

Evoluția către optimul global al problemei (care din păcate nu este în mod garantat atins, dar există o probabilitate nenulă să fie obținut) se face datorită presiunii de selecție.

În sens larg, algoritmii genetici sunt acele modele de calcul bazate pe o populație de soluții, care folosesc selecția și operatori de recombinare pentru a genera noi puncte în spațiul de căutare.

Prima formulare în acest domeniu a fost făcută de John Holland(1975), care a propus un algoritm cunoscut astăzi sub numele de **algoritm genetic canonic**. Acesta conține în cazul maximizării următorii pași:

1. Se generează **populația inițială**, fiecare individ fiind reprezentat de un cromozom alcătuit dintr-o secvență de L biți (aleși inițial, de obicei aleator).

2. Se **evaluatează** populația curentă calculnd funcția obiectiv f_i pentru fiecare individ.
3. Se efectuează **selectia** și se determină o populație intermediară. În acest scop se calculează pentru fiecare individ parametrul f_i/f_{med} ("fitness value"), prin raportări la valoarea medie pe populație a funcției obiectiv. Posibilitatea ca un individ să fie selectat este proporțională cu acest parametru.
4. În populația intermediară se aplică operatorul de **încrucișare**. Se alege aleator o pereche de indivizi și cu probabilitatea p_c se recombina cei doi noi indivizi care sunt inserați în generația următoare a populației. Pasul se repetă pñă când populația intermediară are dimensiunea populației inițiale.
5. Se aplică operatorul **de mutație** la întreaga populație nouă. Fiecare bit al fiecărui individ este schimbat cu o probabilitate p_m mică (de obicei sub 1%).
6. Dacă nu este îndeplinit criteriul de oprire se reia pasul 2, considernd ca populație curentă noua generație, obținută în urma pasului 5.

Un algoritm genetic canonic este caracterizat de următoarele alegeri:

- dimensiunea L a unui cromozom și felul în care este codificată o potențială soluție;
- dimensiunea N a populației și felul în care este aceasta inițiată;
- modul de definire a funcției de adecvare și mecanismul de selecție ales;
- mecanismul de încrucișare și probabilitatea de încrucișare p_c ;
- mecanismul de mutație și probabilitatea de mutație p_m ;
- criteriul de oprire. De obicei funcția de adecvare a unui individ f_i este egală cu funcția obiectiv a problemei de maxim, presupunnd că această este pozitivă. Dacă nu, funcția obiectiv este supusă unei transformări (de exemplu inversare).

Modul uzual de **codificare** a unei soluții potențiale este cea binară. Considernd o problemă de optimizare cu n variabile, cu restricții de tip frontieră, (eventual alese convențional), spațiul paralelipipedic de căutare va putea fi parametrizat folosind n numere întregi. Valoarea maximă a fiecărei coordonate discrete (întregi) depinde de ct de precis se dorește să se rezolve problema.

- În reprezentarea **binară naturală**, cromozomul unui individ se construiește înlănțuind biții corespunzător fiecărei coordonate întregi reprezentată în baza 2. Biții care corespund unei variabile alcătuiesc o "genă" în cadrul cromozomului. În consecință,

dimensiunea L a unui cromozom depinde de numărul de variabile cu care este calculată soluția problemei de optimizare, fiind o măsură a dificultății problemei ("dimensiunea" spațiului de căutare discretă, exprimată în număr de soluții discrete este 2^L). Dezavantajul codificării binare naturale constă în faptul că biții din cromozom au ponderi diferite, de exemplu modificarea bitului cel mai semnificativ dintr-o genă a cromozomului (genotip) produce o modificare mult mai importantă asupra soluției (fenotipului) decât cea produsă de modificarea bitului cel mai puțin semnificativ.

- O altă variantă de codificare care elimină acest dezavantaj, este cea bazată pe **codul Gray**, care are proprietatea că modificarea oricărui bit (în genotip) determină o modificare minimă a soluției (fenotipul).

De obicei dimensiunea N a populației este aleasă între 10 și 100 (corelată de obicei cu dimensiunea L a cromozomului) iar inițializarea se recomandă să fie făcută astfel încât populația să aibă o diversitate ct mai mare.

Inițializarea populației se poate realiza în două moduri în principiu diferite:

- **inițializarea deterministă**, de exemplu în cele 2^n vrfuri ale paralelipipedului de căutare, sau în interiorul paralelipipedului, în nodurile unei rețele uniforme ce discretizează domeniul de căutare;
- **inițializarea stocastică**, bazată pe împrăștierea aleatoare a indivizilor în domeniul de căutare, cu o distribuție uniformă în volum și/sau pe frontiera paralelipipedului.

Cunoștințe apriorice referitoare la zona în care sunt șanse mari să se afle soluția pot fi utilizate cu succes la inițializare. De exemplu, împrăștierea aleatoare poate fi făcută cu o distribuție normală (Gauss) în jurul unui punct, în care este posibil să se afle soluția.

Sunt cunoscute mai multe mecanisme clasice pentru realizarea **selecției**.

În metoda **resturilor stocastice**, acei indivizi pentru care $f_i/f_{med} > 1$, sunt copiați în $n = [f_i/f_{med}]$ exemplare, urmnd ca apoi fiecare individ să fie plasat în populația intermediară cu o probabilitate egală cu partea fracționară a raportului f_i/f_{med} .

În mecanismul **rețelei**, numit de **alegere stocastică cu înlocuire** se folosește modelul unei roți de ruletă la periferia căreia fiecare individ ocupă un spațiu proporțional cu valoarea sa f_i . Prin învrtirea repetată a ruletei (alegerea aleatoare a unui punct de la periferia roții) se alege cte un individ care va fi plasat în populația intermediară.

Modelul de **alegere stocastică universală** se bazează pe o roată de ruletă cu N pointeri echidistanți, ce se învrtesc deasupra unui disc circular cu felii de mărime proporționale cu valorile indivizilor. La o singură rotire a ruletei se selectează cei N indivizi ai populației intermediare.

Metodele prezentate sunt bazate pe valoarea funcției de adecvare f_i ("fitness") a unui individ, motiv pentru care sunt numite scheme de selecție proporțională. În prezent sunt tot mai des folosite metodele bazate pe ordinea indivizilor și nu pe valoarea lor. O selecție de acest tip este cea bazată pe **turneu eliminator** în care se face o selecție aleatoare și este introdus în populația intermediară cel mai bun individ. Tot bazat pe ordine este și "clasamentul liniar", în care probabilitatea unui individ de a fi selectat este proporțională cu numărul de ordine în clasament.

Operatorul de încrucișare poate fi implementat în mai multe feluri. Mecanismul cel mai simplu este **încrucișarea într-un punct**, în care se alege aleator un punct de încrucișare cuprins între 1 și L și cei doi cromozomi se fragmentează în acel punct iar segmentele se schimbă între ele.

Încrucișarea în două puncte presupune alegerea aleatoare a două numere cuprinse între 1 și L , urmnd ca cei doi cromozomi să schimbe segmentul cuprins între aceste două puncte. Cromozomii se comportă în acest caz ca și cum ar avea structură inelară.

În **încrucișarea uniformă** fiecare bit este independent de alți biți și de fapt nu există nici o legătură între biți. Pentru fiecare bit de la 1 la L al urmașului se alege aleator un bit de pe poziția respectivă de la unul din părinți.

Nu toate perechile de indivizi selectate din populație sunt supuse încrucișării. De obicei împerecherea se efectuează cu o probabilitate p_c cuprinsă între 60% și 100 %. Perechile de părinți neîncrucișați sunt transferate în populația intermediară, astfel existnd șansa ca cel mai bun individ dintr-o populație să supraviețuiască neperturbat în generația următoare.

Dacă indivizii dintr-o subpopulație (sau întreaga populație) au aceeași biți pe anumite poziții, atunci aceștia vor fi moșteniți în urma încrucișării. Un astfel de șablon, ca de exemplu 10 * * * 01 * * se numește "o schemă" (în cazul nostru, de ordinul 4, deoarece conține 4 biți fixați urmnd ca cei marcați cu * să poată fi 0 sau 1 în spațiul de căutare, care poate fi privit ca un hiperplan (de dimensiune 9 în acest exemplu) și ea definește un "hiperplan" (de dimensiune 5 în acest exemplu).

Prin încrucișare este explorată rețeaua genetică a populației iar prin selecție sunt eliminate progresiv genele necorespunzătoare, populația suferind un proces de convergență către un individ cu o comportare (adecvare) mai bună.

Este posibil ca totuși după un număr redus de generații, procesul de selecție să conducă la o populație alcătuită din indivizi identici (biții de pe o poziție ai tuturor indivizilor au aceeași valoare). O astfel de populație se spune că este degenerată. Dacă acest lucru se întâmplă fără ca soluția obținută să fie satisfăcătoare, se spune că algoritmul genetic a avut o **convergență prematură**.

Acest lucru se datorează pierderii diversității genetice și se poate întâmpla cnd populația

are dimensiuni prea mici.

Pentru a restaura **diversitatea genetică**, operatorul de mutație joacă un rol esențial. Mutațiile întrerup continuitatea procesului de convergență, generat de încrucișări și selecții succesive, dar asigurând diversitatea genetică ele reprezintă un "rău necesar" pentru a obține convergența globală. Prin caracterul lor aleator ele pot determina "ieșirea din groapa" unui minim local ("hill-climbing") și pot elimina în acest fel convergența prematură. Din acest motiv unii cercetători consideră că încrucișările nu sunt neapărat necesare și că mutațiile singure pot fi suficiente pentru genera metode de căutare robuste și eficiente. Cu toate acestea probabilitatea de aplicare a mutației este de ordin foarte mic $p_m = 0.1\% \div 5\%$.

O altă cauză a convergenței premature o poate constitui tendința populației de a degenera, adică de a deveni o populație cu indivizi foarte asemănători. O metodă de sporire a presiunii de selecție¹ constă în **scalarea funcției de adecvare** prin micșorarea valorii corespunzătoare individului cel mai rău plasat și creșterea celei corespunzătoare celui mai bun individ.

Oprirea procesului de evoluție are loc de obicei după un număr limită de generații. În unele cazuri generațiile sunt numărate de la ultima modificare a celui mai bun individ.

Dintre codurile simple de algoritmi genetici menționăm **GENESIS** și **ES GAT** (Elitist Simple Genetic Algorithm with Tournament Selection).

E.2 Strategii evoluționiste

În afară de algoritmi genetici canonici s-au dezvoltat și alte modele evoluționiste de calcul, dintre care menționăm **Strategiile evoluționiste (ES)** (Rechenberg-1973, Schwefel-1975), cu următoarele două variante:

- În varianta $(\mu + \lambda) - ES$ cei μ părinți produc λ urmași și populația este redusă din nou la μ părinți prin selecția celor mai buni indivizi din populația comună formată din părinți și urmași. În acest fel, un părinte supraviețuiește pînă cînd este înlocuit de un urmaș mai bun. Deoarece nu apare o generație intermediară, acumularea cromozomilor îmbunătățiți în populație ca și creșterea celei mai bune performanțe are un caracter monoton.

¹Presiunea de selecție se referă la gradul în care indivizii buni sunt favorizați: cu cât presiunea de selecție este mai mare, cu atât mai mult sunt favorizați indivizii mai buni să devină părinți. Rata de convergență a unui algoritm evoluționist este determinată în mare măsură de presiunea de selecție: cu cât aceasta este mai mare, cu atât rata de convergență crește. Dacă presiunea de selecție este prea mare, algoritmul ar putea converge către puncte sub-optime (extreme locale).

- Varianta $(\mu, \lambda) - ES$ este conceptual apropiată algoritmilor genetici clasici, deoarece urmașii înlocuiesc părinții și apoi are loc selecția. Spre deosebire de GA canonic, **operatorii de recombinare** în ES sunt diferiți de încrucișarea binară clasică, de exemplu parametrii urmașilor se pot obține prin media parametrilor părinților (aceasta permite utilizarea unor codificări ”continui”, prin numere reale, ale parametrilor).

De obicei în strategiile evoluționiste codificarea indivizilor se realizează printr-un vector cu componente reale și nu un șir de biți ca în cazul GA. Schemele de selecție (μ, λ) sunt bazate pe ordine și constau în extragerea a μ indivizi și selectarea celor mai buni λ indivizi. Se spune că un algoritm este **elitist**, dacă se colectează și se menține în populație cel mai bun individ găsit până în momentul respectiv. Convergența unui algoritm elitist este monotonă, deoarece la nici o generație nu are loc scăderea funcției de adecvare a celui mai bun individ.

Dintre implementările metodei $(\mu + \lambda)ES$ menționăm programul **Genitor** (Whitley 1988), care față de GA canonic are următoarele trei deosebiri importante:

- în urma reproducerii, cei doi părinți generează un singur urmaș, care este plasat imediat înapoi în populație,
- urmașul nu înlocuiește părinții, ci cel mai neadaptat membru al populației,
- pentru a menține o presiune constantă de selecție pe parcursul căutării, populația este împărțită în clase, reproducerea fiind permisă doar claselor superioare.

Un alt algoritm genetic elitist este implementat în programul **CHC**, dezvoltat de L. Eshelman în 1991. Principalele caracteristici ale acestui algoritm de tip $(\mu + \lambda)ES$ sunt:

- după recombinare, cei mai buni N indivizi distincți sunt selectați (**eliminnd duplicatele**) din populația de părinți și urmași, pentru a forma noua generație (se creează în acest fel o presiune de selecție suficientă pentru a nu mai fi necesar alt mecanism de selecție);
- cei doi părinți selectați aleator trebuie să se afle la o distanță Hamming suficient de mare unul față de altul (**”prevenirea incestului”** este destinată promovării diversității genetice), iar recombinarea se realizează cu un operator de încrucișare uniformă în care jumătate din biții diferiți sunt luați de la un părinte și restul de la celălalt;
- este utilizată o populație redusă (de cca 50 indivizi) iar cnd aceasta își pierde diversitatea are loc o **mutație catastrofală** (la toți indivizii, cu excepția celui mai bun), urmnd ca după mutație să fie repornită căutarea genetică folosind numai încrucișări.

Un program care folosește codificarea prin valori reale și nu binare, precum și operatori genetici de recombinare bazați pe mediere și nu pe încrucișare binară este **Genocop**, dezvoltat de Michalewicz în 1992.

O altă variantă de algoritmi bazați pe evoluție este cea a **algoritmilor hibridi (mimetici)** care permite indivizilor ca pe lângă operațiile stocastic-evoluționiste specifice să se efectueze și o căutare determinist-locală a optimului. Astfel, după ce fiecare urmaș este creat, acesta constituie o inițializare pentru un algoritm de tip "local hill-climbing", realizat de obicei cu tehnici deterministe sau prin căutare aleatoare. Cromozomii îmbunătățiți prin această "evoluție Lamarckiană" sunt plasați în populație pentru a participa la competiția pentru reproducere. În multe probleme de optimizare, algoritmii hibridi obțin rezultate superioare (din punctul de vedere al robusteții și eficienței) față de algoritmii tradiționali. Avantajul acestor abordări constă și în dimensiunea mică a populației necesare. Cu doar 20 de indivizi se poate realiza o căutare suficient de robustă.

E.3 Prototipul unui algoritm evoluționist

După cum se cunoaște, algoritmii genetici (GA) și strategiile evoluționiste (ES) au multe caracteristici comune, ambele abordări făcând parte dintr-o clasă mai largă de algoritmi, numită algoritmi evoluționiști (EA), sau uneori algoritmi genetici generali.

Prototipul unui algoritm evoluționist simplu (SEA) conține următoarele etape:

1. Inițializarea populației
2. Evaluarea populației gradului de adecvare
3. Verificarea criteriului de oprire
4. Generarea tabloului de împerechere și schimbare
5. Mutația
6. Se reia algoritmul de la etapa 2

Pentru a codifica posibilele soluții se utilizează cromozomi (alcătuiți de obicei din șiruri de biți grupați în blocuri constitutive). Un parametru important al algoritmului îl constituie dimensiunea populației. Dacă aceasta este prea mică, numărul inițial de blocuri constitutive este prea mic și algoritmul riscă să convergă spre o soluție suboptimală (un extrem local). Dacă populația este prea mare algoritmul irosește timp prin procesarea inutilă a unor indivizi asemănători.

În etapa a doua a algoritmului se evaluează funcția obiectiv a fiecărui individ și pe baza ei se determină gradul de adecvare (fitness) al indivizilor. Din diferite motive, gradul de adecvare nu reflectă perfect funcția obiectiv, și se spune că evaluarea se realizează cu zgomot (fie că funcția obiectiv este evaluată rapid, dar aproximativ, fie că gradul de adecvare este determinat aproximativ prin eșantionare). Acest zgomot influențează decizia în procesul de selecție. Din cauza lui, în cazul schemelor de selecție bazate pe ordine, ca turneul, clasamentul liniar sau selecția ($\lambda\mu$) viteza de convergență a algoritmului scade. În cazul schemelor de selecție proporțională, zgomotul nu modifică viteza globală de convergență, în schimb aceste mecanisme au alte probleme (scalarea și efortul mare de calcul).

În algoritmi evoluționiști se utilizează diferite criterii de oprire. O metodă este de a impune numărul total de generații sau o limită a timpului de execuție. Algoritmul poate fi oprit dacă cea mai bună soluție nu se modifică un număr dat de generații. O altă metodă se bazează pe convergența populației către aproape aceleași valori a funcției obiectiv sau gruparea întregii populații în jurul unui individ, eventual convergența populației către un singur individ, dacă se elimină de la un moment dat mutația. Un exemplu tipic de algoritm evoluționist este implementat în programul **mGA** (Messy Genetic Algorithm), care urmărește rezolvarea ct mai rapidă a problemelor dificile. S-au obținut complexități subpătratică $O(L \log(L))$ în funcție de numărul variabilelor, pentru probleme codificabile cu 30 până la 150 biți. Spre deosebire de alte programe în mGA se utilizează o reprezentare compactă a legăturilor specifice blocurilor constructive ale cromozomului. Un cromozom este alcătuit dintr-o mulțime de gene (caracterizată fiecare prin nume și valoare), care pot fi subdeterminate (lipsesc unele gene) sau supradeterminate (unele gene sunt în contradicție. mGA are două etape, în prima, numită faza primordială, se selectează populația pornind de la o inițializare parțial enumerativă (sau complet probabilistică în versiunile mai noi, urmată de o filtrare a blocurilor constitutive). A doua etapă, numită fază de juxtapunere, este similară unui algoritm genetic fără mutație, la care operatorul de încrucișare este înlocuit cu un operator de "tăiere și altoire".

Trebuie remarcat că lungimea șirului de gene crește pe măsura procesării.

E.4 Algoritmi cu nișe pentru optimizarea funcțiilor multimodale

Algoritmi tradiționali prezentați anterior nu sunt potriviți pentru optimizarea funcțiilor multimodale, care au mai multe extreme globale, deoarece întreaga populație are tendința să evolueze către un singur punct, maxim global al funcției obiectiv. În multe cazuri practice interesează găsirea optimelor multiple, fie globale fie locale. Pentru rezolvarea acestui tip de probleme, sunt folosiți algoritmi genetici care folosesc conceptul de nișă.

Algoritmii genetici cu nișe permit formarea mai multor subpopulații stabile asociate "nișelor" aflate în vecinătatea soluțiilor optime. Acești algoritmi mențin diversitatea populației, prevenind căderea în capcanele unor extreme locale și permițând investigarea a mai multor "vrfuri" în paralel. Mecanismul de nișe modelează ecosistemele naturale în care mai multe specii coexistă și evoluează în paralel, fiecare găsiindu-și propria nișă în mediul înconjurător. O specie este definită ca un grup de indivizi cu caracteristici similare, capabili să se împerecheze reciproc dar incapabili să se împerecheze cu alte specii. În fiecare nișă resursele mediului sunt finite și trebuie împărțite între indivizii unei specii care ocupă acea nișă.

Pentru a putea implementa conceptul de nișă, în algoritm se alege un număr de selecții specific.

Dintre algoritmii genetici cu nișe, cei mai cunoscuți sunt cei bazați pe **vecinătăți explicite**, ca de exemplu cei cu "partajarea adecvării" (fitness sharing) și cei bazați pe **vecinătăți implicite**, ca de exemplu cei cu "aglomerare" (crowding).

În **algoritmi cu partajarea adecvării** gradul de adaptare (fitness) este considerat ca o resursă a nișei și este în consecință diminuat de un număr de ori aproximativ egal cu numărul de indivizi din subpopulația nișei respective:

$$f'_i = f_i/m_i,$$

în care

$$m_i = \sum_{j=1}^N s(d_{ij})$$

unde N reprezintă dimensiunea populației, d_{ij} reprezintă distanța dintre indivizii i și j , iar s este o funcție de partajare (sharing), care măsoară similitudinea dintre doi indivizi:

$$s(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_s)^\alpha & \text{dacă } d < \sigma_s; \\ 0 & \text{în rest,} \end{cases}$$

întorcnd zero dacă indivizii sunt mai depărtați decât raza nișei σ_s și o valoare cu att mai apropiată de 1 cu ct distanța dintre indivizi este mai mică. Coeficientul α se alege de obicei 1 (funcție de partajare triunghiulară) dar poate fi ales și 1/2 sau 2 (în cazul funcției parabolice).

Distanța între indivizi este de obicei norma euclidiană a diferenței reale, din spațiul de căutare (similitudinea fenotipică) și mai rar evaluată ca distanța Hamming între cromozomii binari (similitudine genotipică).

O tehnică uzuală pentru a îmbunătăți eficiența împărțirii este de a rescala gradul de adecvare folosind parametrul $\beta > 1$:

$$f''_i = f_i^\beta / m_i.$$

Dacă β este totuși prea mare, predominanța superindivizilor poate determina convergența prematură, motiv pentru care se recomandă creșterea sa progresivă în timpul căutării ("parameter annealing").

Încrucișarea dintre indivizi aflați în nișe diferite conduce de obicei la apariția unor urmași neperformanți, "indivizi letali". Pentru a evita acest lucru se aplică cu succes scheme de **restricție a împerecherii** ("mating restriction").

Într-o schemă tipică de restricție a împerecherii, primul părinte este ales aleator (sau în urma unui terneu), iar al doilea este ales astfel încât să satisfacă o restricție impusă distanței față de primul părinte. Alegerea se poate face din întreaga populație, caz în care complexitatea are ordinul $O(N^2)$ sau dintr-o parte a ei, de dimensiune MF (factor de împerechere), caz în care complexitatea calculului are ordinul $O(MF \times N)$. Restricția impusă distanței se poate baza pe o inegalitate de forma $d < \sigma_m$, în care σ_m este "raza de împerechere", iar dacă nu există un individ care să satisfacă această inegalitate al doilea părinte se alege aleator din întreaga (sub)populație sau individul cel mai apropiat. Dezavantajul acestei metode constă în faptul că pentru indivizi aflați la frontiera unei nișe, perechea sa are 50% șansa să se afle în afara nișei. Acest dezavantaj este eliminat dacă se utilizează reproducerea în linie ("line breeding"), bazată pe alegerea celui de-al doilea părinte, ca fiind "campionul nișei". În tehnica numită de "reproducere internă și intermitent externă" ("inbreeding with intermitent cross-breeding"), înmulțirea are loc în interiorul nișei, att timp ct gradul mediu de adecvare al subpopulației nișei crește și cu indivizi din nișe diferite, dacă adecvarea medie este staționară.

Pentru a elimina aceste dezavantaje s-au propus multe alte alternative la metodele prezentate, dintre care menționăm: nișe secvențiale, nișe dinamice, sisteme cu imunitate, specii cu coevoluție, algoritmi genetici și mecanisme bazate pe aglomerare.

O metodă interesantă de **nișe adaptive obținute prin coevoluție** este CSN (Co-evolutionary Shared Niching). Tehnica este inspirată din modelul economic al competiției monopoliste, în care capitaliștii își plasează geografic afacerile astfel încât să-și maximizeze profitul, obținut de la o populație de clienți. Algoritmul menține două populații, una de "afaceriști" și alta de "clienți", prima indicnd nișele iar a doua posibilele soluții ale problemei. Cele două populații urmăresc maximizarea separată a intereselor lor, permițnd stabilirea att a locațiilor cât și a razelor nișelor, adaptate unui peisaj complex.

Fie C populația de clienți cu mărimea $n_c = ||C||$ și B populația de afaceriști cu $n_b = ||B||$, se spune că un client aparține afacerii b dacă distanța Hamming $|b - c|$ este minimă. Mulțimea clienților unei afaceri se notează cu C_b iar numărul lor cu $m_b = ||C_b||$.

Funcția de adecvarea a unui client se definește:

$$f'(c) = f(c)/m_b|_{c \in C_b}$$

prin împărțirea gradului brut de adecvare la numărul de clienți din aria de serviciu a afacerii la care individul este client. Afacerile vor fi plasate în vrfurile nișelor, dar nu la distanța mai mică decât d_{min} una de alta.

Se consideră gradul de succes al unei afaceri (fitness) egal cu suma gradului brut de adecvare a clienților ei:

$$\phi(b) = \sum_{c \in C_b} \bar{f}(c).$$

Afacerea i va fi de preferat afacerii j dacă $\phi_i > \phi_j$, adică dacă $\bar{f}_i > n_j \bar{f}_j / m_i$, deci numărul de clienți ai unei afaceri bune va tinde să fie mai mare decât numărul de clienți ai unei afaceri proaste. După cum se constată, scopul algoritmului CSN este de a determina într-o manieră autoadaptivă poziția și dimensiunea nișelor (care sunt poliedrele Voronoi în jurul afacerilor). În varianta simplă a programului CSN populația de clienți nu are mutații ($p_m = 0$) iar cea de afaceriști nu are încrucișări ($p_c = 0$). Se procesează pe rnd cele două populații folosind încrucișarea simplă și selecția stocastică universală (SUS). O nouă afacere înlocuiește o afacere veche doar dacă este mai bună ca aceasta și este îndeplinit criteriul d_{min} .

Rezultate mai bune se obțin aplicnd mecanismul de imprimare ("inprint"), care permite convertirea celor mai buni clienți în oameni de afaceri.

Metodele de aglomerare aplicate în algoritmi genetici par cele mai promitătoare tehnici de nișe cu vecinătăți implicite și ele se bazează pe modul în care noile elemente sunt inserate în populație, prin înlocuirea elementelor similare.

Aglomerarea clasică a fost propusă de DeJong încă din 1975, și se bazează pe faptul că numai o parte G ("generation gap") din populație se reproduce și moare la fiecare generație, iar un urmaș înlocuiește individul cel mai apropiat (cu maximă similitudine genotipică), ales dintr-o subpopulație formată din CF (factor de aglomerare) indivizi ai populației globale. Pentru valori mici ale factorului $CF < 30\%$ au loc multe erori de înlocuire, ceea ce face metoda ineficientă, iar pentru valori mari ale factorului CF , efortul de comparație este mare, deoarece are ordinul $O(CF \times N)$.

O metodă eficientă este cea a **aglomerării deterministe (DC)**, care se bazează pe competiția între urmașii și părinții din aceeași nișe. După încrucișare și eventuala mutație, urmașul înlocuiește cel mai apropiat părinte (în sens fenotipic), dacă este mai bun ca acesta. Câștigătorul este determinat în urma unui turneu în două seturi (P1 contra U1 și P2 contra U2 în primul set și P1contra U2 și P2 contra U1 în al doilea set). Complexitatea acestui algoritm este doar liniară, avnd ordinul $O(n)$.

O altă variantă de nișe bazate pe aglomerare este **selecția cu turneu restrns (RTS)**. Aceasta este similară aglomerării clasice, cu deosebirea că inserția se face după o competiție (urmașul înlocuiește individul similar, doar dacă este mai bun ca acesta).

Metoda RTS necesită un efort de calcul mai mare decât metoda DC, în schimb rezultatele sunt în general ceva mai bune.

E.5 Algoritmi evoluționiști paraleli

Algoritmii evoluționiști sunt utilizați pentru rezolvarea unor probleme de mare complexitate, care, în consecință, au un timp mare de calcul. Pentru a reduce timpul până se obține soluția dorită, o metodă constă în utilizarea în paralel a mai multor calculatoare. Utilizând modelul biologic este clar că evoluția genetică are loc "în paralel". Atunci când se dezvoltă un algoritm paralel sau distribuit pe mai multe sisteme de calcul este necesar să se reducă la minim comunicarea între procesoare, deoarece aceasta poate necesita un timp suplimentar, mai mare decât timpul propriu-zis de calcul.

Metoda naturală de paralelizare constă în distribuirea populației pe mai multe procesoare.

Cel mai simplu model de algoritm genetic paralel global este **modelul "fermier-muncitori"**, în care procesorul master (server) colectează valorile funcțiilor obiectiv, calculează gradul de adecvare și efectuează selecția pe când procesoarele sclavi (clienți) întrețin fiecare câte o subpopulație (primită de la server) aplicând operatorii genetici și evaluând subpopulația rezultată.

Un alt algoritm genetic paralel este cel bazat pe **modelul insulelor** (Whitley 1990). Populația este împărțită într-un număr de subpopulații egal cu numărul de procesoare disponibile. Fiecare procesor rulează un algoritm genetic canonic, Genitor sau CHC. Periodic (de exemplu la cinci generații), câteva copii (clone) ale indivizilor din fiecare subpopulație migrează către alte subpopulații cu scopul de a transporta material genetic proaspăt spre alte "insule", pe care se află subpopulații cu evoluții independente.

Prin introducerea migrațiilor modelul insulelor este capabil să exploateze diferența dintre diversele subpopulații, ca o sursă de diversitate genetică, fără ca fiecare populație să trebuiască să fie de mari dimensiuni. Frecvența și numărul migrațiilor nu trebuie să fie prea mare pentru a evita riscul de convergență prematură.

Un program paralel este **iiGA** (Injection Island GA), bazat pe modelul insulelor, cu migrări asincrone și admitând atât topologie statică cât și dinamică (ierarhică). Fiecare insulă având lungimi diferite pentru cromozomi. În acest fel aceeași problemă este codificată cu nivele diferite de rezoluție, iar migrarea are loc numai de la noduri de rezoluție joasă la noduri cu rezoluție înaltă (prin injecție).

Dintre codurile paralele menționăm **GALOPPS** dezvoltate de Michigan State University - Genetic Algorithm Research and Appl. Group (MSU-GARAGE), precum și

sistemul **PeGAsuS** (Programming Env. for Parallel genetic Algorithms), dezvoltat la GMD.

E.6 Analiza parametrilor algoritmilor evoluționiști

Un parametru important al unui algoritm genetic este dimensiunea populației. În cazul populațiilor mici algoritmi genetici fac erori de decizie, iar calitatea convergenței este slabă, soluția fiind condusă fie de mutațiile arbitrare fie suferind o convergență prematură.

În cazul populațiilor mari, algoritmi pot discrimina în mod robust între blocurile constitutive adecvate și cele neadecvate, astfel încât recombinația în paralel a acestor blocuri conduce relativ rapid la soluție, chiar în cazul problemelor dificile. Din păcate, dimensiuni prea mari ale populației necesită un efort inacceptabil de calcul. Dimensiunea optimă este dependentă de sursele de zgomot stohastic, cum sunt zgomotul de selecție, zgomotul operatorilor genetici și zgomotul din funcția obiectiv:

$$N = 2cK \frac{\sigma_M^2}{d^2},$$

în care

c - este un factor de scalare (cu valori între 2 și 20) ce măsoară nivelul de încredere că populația nu va converge prematur;

$K = \chi^k(M-1)/m$, în care χ este cardinalitatea alfabetului (uzual 2), k este lungimea celui mai mare bloc constructiv iar m este numărul de blocuri constructive din cromozom;

d - este minimul semnalului diferență dintre două blocuri aflate în competiție;

$\sigma_M^2 = \sigma_F^2 + \sigma_N^2$ - dispersia populației plus dispersia zgomotului funcției de adecvare.

S-a demonstrat că numărul de generații necesare convergenței are ordinul $O(\log N)$ pentru schemele de selecție bazate pe ordin (turneu, clasament liniar) și ordinul $O(N \log N)$ pentru scheme de selecție bazate pe valoare (ruleta, selecție stohastică universală). Considernd că dimensiunea populației N are ordinul $O(L)$, în care L este lungimea cromozomului, rezultă că efortul de calcul pentru cele două tipuri de metode de selecție are ordinul $O(L \log L)$ și respectiv $O(L^2 \log L)$.

Anexa F

Inițiere în Ψlab (*Scilab*)

Scilab (Ψlab) este un mediu de programare sub sistemul de operare Unix care permite rezolvarea unor probleme tipice de matematică prin efectuarea de calcule matematice, trasarea de grafice, programare în limbaj specific. El emulează limbajul **MATLAB**, având extensii suplimentare.

Majoritatea temelor prezentate în carte necesită utilizarea acestui mediu de programare. De aceea, această anexă are ca scop familiarizarea cu *Scilab*.

F.1 Înainte de toate....

Una din componentele mediului *Scilab* este interpretorul care permite introducerea în mod interactiv a unor comenzi de la consolă și executarea imediată a acestora.

- **Cum se lansează în execuție interpretorul?** Lansarea interpretorului se face prin invocarea numelui său:

```
scilab < ↵ >
```

- **Cum se oprește interpretorul?** La promptul *Scilab* --> se dă comanda:

```
quit < ↵ >
```

- **Există programe demonstrative?** Apăsăți butonul “Demos” și apoi (de exemplu) “Introduction to SCILAB”. Sunt vizualizate instrucțiunile utilizate precum și rezultatele lor.

- **Ajutor !!.** Lista comenzilor și semnificația lor se poate obține apăsând butonul “Help”. Fereastra de help este împărțită în două. În partea de jos sunt principalele capitole. Pentru rezolvarea temelor veți avea nevoie de: “Scilab Programming”, “Graphic

Library”, “ Utilities and Elementary Functions” și “Linear Algebra”. Partea de sus conține comenzi legate de subiectul capitolului selectat. Selectarea unei astfel de comenzi are ca rezultat apariția unei ferestre care descrie sintaxa comenzii, parametrilor ei, modul de apelare.

Există și help-on-line. La promptul *Scilab*-ului dați comanda:

```
help < ↵ >
```

• **Nu știi comanda care face?** O comandă utilă este comanda **apropos**, care caută cuvinte cheie în help-ul *Scilab*-ului.

Exercițiul F.1: a) Folosind comanda **help** aflați care este sintaxa comenzii **apropos**.

b) Folosind comanda **apropos** aflați care sunt comenzile cu care se pot trasa spectre de linii de câmp.

F.2 Variabile și constante.

În rezolvarea temelor veți folosi în exclusivitate matrice cu elemente reale. Un număr poate fi considerat o matrice cu un singur element.

• **Dimensiunea unei matrice** nu trebuie declarată explicit.

Exercițiul F.2: Care este efectul comenzii:

```
--> a(10,5) = 1
```

• **Introducerea unei matrice** se poate face natural astfel:

```
--> A = [ 1  2  3
          4  5  6
          7  8  9 ]
```

Într-o scriere compactă, liniile matricei pot fi separate prin “;” astfel:

```
--> A = [1 2 3; 4 5 6; 7 8 9]
```

Pentru separarea elementelor unei linii se poate folosi caracterul blank (ca mai sus) sau virgula:

```
--> A = [1,2,3;4,5,6;7,8,9]
```

Exercițiul F.3: Știind că vectorii sunt un caz particular de matrice, care sunt comenzile cu care se va introduce un vector linie, respectiv coloană cu elementele 1, 2, 3?

Exercițiul F.4: Comentați următoarea instrucțiune:

```
--> u = 12.4e-3
```

Observație: variabilele utilizate într-o sesiune de lucru ocupă memoria sistemului pe măsură ce sunt definite. Pentru a vizualiza lista variabilelor existente la un moment dat și memoria disponibilă se folosește comanda `who`. Dacă se dorește eliberarea memoriei de una, mai multe sau toate variabilele generate se folosește comanda `clear`.

Exercițiul F.5: Executați instrucțiunea `who`. Ce reprezintă `%e` `%pi` `%f` `%t` `%eps` `%inf`?

F.3 Atribuire și expresii

Instrucțiunea de atribuire are sintaxa:

```
variabila = expresie
```

sau simplu:

```
expresie
```

în care `variabila` este numele unei variabile, iar `expresie` este un șir de operatori și operanzi care respectă anumite reguli sintactice. În a doua formă, după evaluarea expresiei, rezultatul este atribuit variabilei predefinite `ans`.

• **Operatorii aritmetici**¹ recunoscuți de *Scilab* sunt:

- + adunare;
- scădere;
- * înmulțire;
- / împărțire la dreapta;
- \ împărțire la stânga;
- ^ ridicare la putere.

Pentru transpunerea unei matrice se folosește operatorul “apostrof” ca în exemplul:

```
--> B = A'
```

¹Operatorii aritmetici se aplică unor operanzi aritmetici, iar rezultatul este aritmetic.

în care matricea B se calculează ca transpusa matricei A.

Observații:

1. Adunarea și scăderea pot fi efectuate:

- între două matrice cu aceleași dimensiuni;
- între o matrice și un număr (caz în care numărul este adunat, respectiv scăzut din fiecare din elementele matricei).

2. Înmulțirea poate fi efectuată:

- între două matrice dacă lungimea liniei primei matrice este egală cu lungimea coloanei celei de a doua matrice;
- între un număr și o matrice (caz în care numărul este înmulțit cu fiecare din elementele matricei);
- între două matrice cu aceleași dimensiuni (element cu element), caz în care operatorul `*` este precedat de un punct, ca în exemplul:

```
--> C = A .* B
```

3. Împărțirea matricelor poate fi făcută în mai multe feluri:

- la dreapta (pentru matrice pătrate și nesingulare):

```
--> X = B / A
```

echivalent cu:

```
--> X = B * inv(A)
```

sau cu:

```
--> X = B * A^(-1)
```

- la stânga:

```
--> X = A \ B
```

echivalent cu:

```
--> X = inv(A) * B
```

sau cu:

```
--> X = A^(-1) * B
```

Dacă A este o matrice dreptunghiulară de dimensiuni $m \times n$, iar b este un vector coloană cu m elemente, atunci împărțirea la stânga $x = A \setminus b$ calculează soluția ecuației $Ax = b$ în sensul celor mai mici pătrate.

- împărțirea unei matrice la un număr (să îl notăm cu u):

$$\rightarrow Y = A / u$$

- împărțirea element cu element a matricelor de dimensiuni egale:

$$\rightarrow C = A ./ B$$

sau

$$\rightarrow C = A ./ B$$

4. Ridicarea la putere A^p se face astfel ²:

- pentru p întreg pozitiv: dacă matricea A este pătrată atunci A se înmulțește cu ea însăși de p ori; dacă matricea A este dreptunghiulară atunci se ridică la puterea p fiecare element din matricea A
- pentru p întreg negativ: dacă matricea A este pătrată atunci inversa ei se înmulțește cu ea însăși de $-p$ ori; dacă matricea A este dreptunghiulară atunci se ridică la puterea p fiecare element din matricea A
- pentru p număr real (dar nu întreg) pozitiv: dacă matricea A este pătrată atunci calculul se face cu ajutorul vectorilor și valorilor proprii ale matricei; dacă matricea A este dreptunghiulară, atunci se ridică la puterea p fiecare element din matricea A .
- pentru p număr real (dar nu întreg) negativ: dacă matricea A este pătrată atunci calculul se face cu ajutorul vectorilor și valorilor proprii ale inversei matricei; dacă matricea A este dreptunghiulară, atunci se ridică la puterea p fiecare element din matricea A .

- **Operatorii de relație**³ recunoscuți de *Scilab* sunt:

< mai mic decât;
 <= mai mic sau egal cu;
 > mai mare decât;
 >= mai mare sau egal cu;
 == egal cu;
 ~= diferit de.

²Nu sunt descrise toate situațiile posibile.

³Operatorii de relație se aplică unor operanzi aritmetici iar rezultatul este logic.

Aceștia permit testarea unor condiții, rezultatul având valoarea F (fals) sau T (adevărat). Dacă operandii sunt matrice de dimensiuni egale, atunci operațiile logice se fac între elementele de pe aceleași poziții, iar rezultatul este o matrice cu elementele F și T.

• **Operatorii logici**⁴ recunoscuți de **Scilab** sunt:

- & conjuncția logică;
- | disjuncția logică;
- ~ negația logică.

Dacă operandii sunt matrice (logice) cu aceleași dimensiuni, atunci operația se face element cu element. Dacă unul din operandi este o valoare logică, atunci acesta se combină cu fiecare din elementele celuilalt operand. Alte situații nu sunt permise.

• **Funcții elementare.** Operandii unor expresii pot fi și apeluri de funcții elementare (de exemplu trigonometrice), sau alte funcții cunoscute. Aceste funcții aplicate unei matrice acționează asupra fiecărui element în mod independent. Lista lor poate fi inspectată apăsând butonul “Help” și apoi “Utilities and Elementary Functions”.

Exercițiul F.6:

Fie $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$ și $v = \begin{bmatrix} 4 & 3 \end{bmatrix}$. Care sunt comenzile **Scilab** pentru rezolvarea ecuației: $A(v^T + x) = b$

F.4 Generarea vectorilor și matricelor

• **Vectorii ai căror elemente formează o progresie aritmetică** pot fi generați cu construcția:

valin : pas : valfin

Exercițiul F.7: Comentați rezultatele următoarelor instrucțiuni:

```
--> x = 1:10
--> y = 1:2:10
--> z = 1:3:10
--> t = 1:-3:10
--> w = -1:-3:-10
--> u = -1:-10
--> v = -10:-1
--> a = 10:-2:-3
```

⁴Operatorii logici se aplică unor operandi logici iar rezultatul este logic.

• **Vectorii ai căror elemente formează o progresie geometrică** pot fi generați cu construcția:

```
logspace(d1, d2, n)
```

Exercițiul F.8: a) Care este semnificația mărimilor $d1$, $d2$, n din comanda `logspace`?

b) Ce generează comanda `linspace` ?

• **Descrierea vectorilor și matricelor pe blocuri.** Vectorii și matricele pot fi descriși pe blocuri, folosind notații de forma:

```
A = [X Y; U V];
```

cu semnificația $A = \begin{bmatrix} X & Y \\ U & V \end{bmatrix}$, în care X, Y, U, V sunt matrice sau vectori.

Exercițiul F.9: Care este rezultatul comenzii:

```
--> A = [1:3 ; 1:2:7]
```

• **Referirea la elementele unei matrice.** Pentru a obține valoarea unui element, se folosesc construcții de forma $a(1, 1)$, $a(1, 2)$. Se pot obține valorile mai multor elemente prin construcții de forma $a(u, v)$ unde u și v sunt vectori. De exemplu $a(2, 1:3)$ reprezintă primele trei elemente din linia a doua a matricei a . Pentru a obține toate elementele liniei 2 se scrie $a(2, :)$.

Exercițiul F.10: Fie $A = \begin{bmatrix} 1 & 10 & 100 & 1000 \\ 2 & 20 & 200 & 2000 \\ 3 & 30 & 300 & 3000 \end{bmatrix}$.

Notați rezultatele și comentați următoarele comenzi:

```
--> A(0,1)
```

```
--> A(2,3)
```

```
--> A(:,3)
```

```
--> A(:, :)
```

```
--> A(3, :)
```

```
--> A(2, 2:4)
```

```
--> A(2:3, 2:4)
```

• **Generarea unor matrice particulare utile** se poate face cu ajutorul funcțiilor:
`eye` matrice cu elementele unitare pe diagonală și nule în rest;

zeros matrice nulă;
ones matrice cu toate elementele unitare;
rand matrice cu elemente aleatoare în intervalul (0,1);
diag construiește o matrice cu o anumită diagonală, sau extrage diagonală dintr-o matrice.

Exercițiul F.11: Comentați următoarele comenzi (unde A este matricea de la exercițiul F.10 iar $v = [1 \ 2 \ 3 \ 4 \ 5]$):

```
--> eye(3)
--> eye(3,3)
--> eye(3,4)
--> eye(A)
--> diag(A)
--> diag(v)
```

Exercițiul F.12: Comentați următoarele comenzi:

```
--> A = diag(1:3)
--> B = [A, eye(A); ones(A) zeros(A)]
--> C = diag(B)
--> D = C'*C
--> E = (D == 14)
```

• *Dimensiunile matricelor (vectorilor)* pot fi modificate în timpul execuției unui program. Pentru a obține dimensiunea unei matrice X se folosește instrucțiunea:

```
[m, n] = size(X)
```

în care m reprezintă numărul de linii și n numărul de coloane. Dimensiunea unui vector v se obține cu:

```
length(v)
```

care are semnificația `max(size(v))`.

Exercițiul F.13: Definiți o matrice oarecare B (de exemplu cu 3 linii și 4 coloane). Executați și comentați următoarele instrucțiuni:

```
--> [m,n] = size(B)
--> B = [B; zeros(1, n)]
--> B = [B zeros(m+1,1)]
```

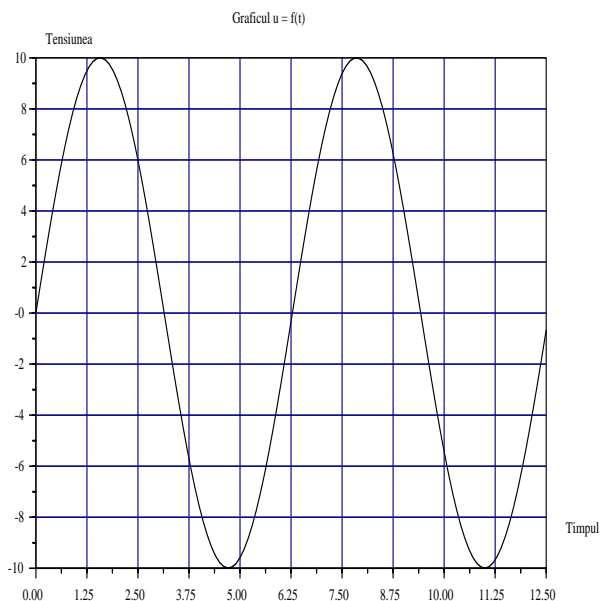



Figura F.1: Acest grafic trebuie obținut la exercițiul F.14

• **Matricea vidă.** *Scilab* operează și cu conceptul de matrice vidă, notată cu `[]` și care este o matrice de dimensiune nulă, fără elemente. Aceasta se dovedește utilă în eliminarea unor linii sau coloane dintr-o matrice dată. De exemplu, instrucțiunea

```
--> B(:, [2 4]) = []
```

are ca efect eliminarea coloanelor 2 și 4 din matricea B . În acest fel, dimensiunea unei matrice poate să și scadă în timpul execuției unui program, nu numai să crească prin adăugarea de noi elemente.

F.5 Instrucțiuni grafice

Funcția principală pentru reprezentări grafice este:

`plot`

Ea are diferite variante⁵, printre care:

⁵Încercați **apropos plot**

```
plot(x,y)
```

în care x este vectorul variabilei independente, iar y este vectorul variabilei dependente. Instrucțiunea:

```
plot(A)
```

în care A este o matrice are ca efect reprezentarea grafică a variației elementelor coloanelor matricei A în funcție de indexul lor. Numărul de grafice este egal cu numărul de coloane.

Funcțiile auxiliare `xtitle` și `xgrid` permit completarea graficului cu un titlu și respectiv adăugarea unui rastru.

Exercițiul F.14: Realizați graficul din figura F.1. El reprezintă funcția $y(t) = 10 * \sin(t)$ pentru $t \in [0, 4\pi]$. Puneți titlul graficului, axelor, și rastrul ca în figură.

F.6 Programare în *Scilab*

Scilab permite utilizarea unor structuri de control (decizii, cicluri, definiri de funcții) ca orice limbaj de programare de nivel înalt. Se pot scrie programe în **Scilab**, ca în orice limbaj de programare.

Avantajul folosirii **Scilab** constă, mai ales, în ușurința cu care se pot postprocesa rezultatele, ținând seama de facilitățile grafice ale sale.

F.6.1 Editarea programelor

Până acum, ați lucrat la consola **Scilab**. Comenzile introduse pot fi scrise într-un fișier și apoi “citite” cu comanda

```
exec
```

Un program în **Scilab** are următoarea structură posibilă:

```
// comentarii
//.....
instrucțiune;    // fără afișarea rezultatului
instrucțiune    // cu afișarea rezultatului
```

Exercițiul F.15: Scrieți comenzile cu care ați rezolvat exercițiul F.14, într-un fișier numit *tema.sci* și apoi executați-l cu comanda:

```
--> exec('tema.sci')
```

Observați ce se întâmplă dacă la sfârșitul fiecărei instrucțiuni adăugați terminatorul “;”.

IMPORTANT: Comenzile necesare rezolvării temelor vor fi scrise în fișiere și executate apoi cu comanda `exec`.

F.6.2 Operații de intrare/ieșire

• Introducerea datelor

Cea mai simplă metodă constă în utilizarea instrucțiunii de atribuire, ca în exemplul:

```
a = 5
```

În cazul unui program scris într-un fișier, este mai convenabil să se folosească funcția `input`. Funcția `input` se utilizează în atribuiri de forma:

```
variabila = input('text')
```

în care “variabila” este numele variabilei a cărei valoare va fi citită de la consolă, iar “text” este un șir de caractere ce va fi afișat, ajutând utilizatorul la identificarea informației ce trebuie introdusă. De exemplu:

```
a = input('Introduceți valoarea variabilei a')
```

• Inspectarea și afișarea rezultatelor

Pentru inspectarea valorilor variabilelor este suficient să fie invocat numele lor:

```
--> x
```

pentru ca interpretorul să afișeze valoarea lor.

Dacă se dorește eliminarea afișării numelui variabilelor din fața valorii sale, atunci se folosește funcția `disp`:

```
--> disp(x)
```

Această funcție poate fi folosită și pentru afișarea textelor, de exemplu:

```
disp('Acest program calculeaza ceva ')
```

Formatul în care sunt afișate valorile numerice poate fi modificat de utilizator cu ajutorul instrucțiunii:

`format`

Operația de ieșire se poate realiza și prin apelul funcției `printf` în instrucțiuni de forma:

```
printf('format',variabile)
```

în care “variabile” sunt variabilele care vor fi scrise în formatul corespunzător instrucțiunii, iar “format” este un șir de caractere ce descrie formatul de afișare. Sunt recunoscute următoarele construcții, similare celor din limbajul C:

`%f` scrierea numărului în format cu virgulă fixă;
`%e` scrierea numărului în format cu exponent;
`%g` scrierea numărului în formatul cel mai potrivit (`%f` sau `%e`).

Celelalte caractere întâlnite în șirul “format” sunt afișate ca atare, de exemplu:

```
printf(' Rezultatul este a = %g', a)
```

Afișarea rezultatelor se poate face și grafic (vezi paragraful F.5).

Exercițiul F.16: Scrieți, într-un fișier, un program prietenos care va permite introducerea de la consola **Scilab** a două numere reale, va calcula suma lor, și va afișa rezultatul în formatul cel mai potrivit.

F.6.3 Structuri de control

• Decizii

Decizia simplă:

Pseudolimbaj	<i>Scilab</i>	Observații
dacă condiție atunci instrucțiuni	if condiție then instrucțiuni end	“condiție” este o expresie care este evaluată, iar dacă rezultatul este adevărat (T), atunci se execută “instrucțiuni”, altfel se execută prima instrucțiune ce urmează după end .

Decizia cu alternativă:

Pseudolimbaj	<i>Scilab</i>	Observații
dacă condiție atunci instrucțiuni1 altfel instrucțiuni2	if condiție then instrucțiuni1 else instrucțiuni2 end	“condiție” este o expresie care este evaluată, iar dacă rezultatul este adevărat (T), atunci se execută “instrucțiuni1”, iar dacă rezultatul este fals (F), se execută “instrucțiuni2”.

Decizia de tip selecție:

Pseudolimbaj	<i>Scilab</i>	Observații
dacă condiție1 atunci instrucțiuni1 altfel dacă condiție2 instrucțiuni2 altfel instrucțiuni3	if condiție1 then instrucțiuni1 elseif condiție2 instrucțiuni2 else instrucțiuni3 end	Pot exista oricâte alternative de selecție.
dacă expresie = expresie1 instrucțiuni1 altfel dacă expresie = expresie2 instrucțiuni2 altfel instrucțiuni	select expresie, case expresie1 then instrucțiuni1, case expresie2 then instrucțiuni2, else instrucțiuni, end	Pot exista oricâte cazuri. “instrucțiuni1” sunt executate dacă expresie==expresie1, etc.

Observație: “instrucțiuni” pot fi scrise după **then**, pe aceeași linie. Cuvântul cheie **then** poate lipsi dacă “instrucțiuni” se scriu pe linie separată⁶.

• Cicluri

Ciclul cu test inițial:

Pseudolimbaj	<i>Scilab</i>	Observații
cât timp condiție instrucțiuni	while condiție instrucțiuni end	Se repetă corpul ciclului, adică “instrucțiuni”, cât timp “condiție” este adevărată. S-ar putea ca “instrucțiuni” să nu fie executate niciodată în timpul rulării programului.

⁶Aceasta este sintaxa *MATLAB*.

Ciclul cu contor:

Ciclul cu contor are două forme, din care a doua este cea generală. Dacă “expresie” este o matrice, atunci “variabila” ia succesiv valorile coloanelor matricei. “instrucțiuni” nu sunt executate niciodată dacă vectorul “valin:pas:valfin” este incorect definit (vid) sau dacă “expresie” este matricea vidă.

Pseudolimbaj	<i>Scilab</i>	Observații
pentru contor = valin, valfin, pas instrucțiuni	for contor = valin : pas : valfin, instrucțiuni end	Forma a doua este cea generală.
	for variabila = expresie, instrucțiuni end	

Ieșirile forțate din cicluri se pot face cu instrucțiunea **break**.

Exercițiul F.17: Scrieți un program care să determine cel mai mare număr întreg n pentru care 10^n poate fi reprezentat în ***Scilab***. Indicație: folosiți pentru testare constanta %inf.

F.6.4 Funcții

Funcțiile sunt proceduri ***Scilab*** (termenii “macro”, “funcție” sau “procedură” au aceeași semnificație).

• Definirea funcțiilor în fișiere

De obicei funcțiile sunt definite în fișiere și “încărcate” în ***Scilab*** cu comanda **getf**. Un fișier conținând o astfel de funcție trebuie să înceapă astfel:

```
function[  $y_1, \dots, y_n$  ] = numefuncție (  $x_1, \dots, x_m$  )
```

unde y_i sunt variabilele de ieșire, calculate în funcție de variabilele de intrare x_j și, eventual, de alte variabile existente în ***Scilab*** în momentul execuției funcției.

Exercițiul F.18: Editați un fișier numit “numefis.sci” cu următorul conținut:

```
function [x,y] = combin(a,b)
    x = a + b
    y = a - b
```

și un fișier numit “main.sci” cu următorul conținut:

```

getf('numefis.sci')
a = input('Introduceti a');
b = input('Introduceti b');
[c,d] = combin(a,b)
printf(' Suma numerelor  a = %g si b = %g este
        a + b = %g',a,b,c);
printf(' Diferenta numerelor a = %g si b = %g
        este a - b  = %g',a,b,d);

```

Executați în **Scilab** comenzile din “main.sci”:

```
--> exec('main.sci');
```

Comentați.

Exercițiul F.19: Scrieți (într-un fișier) o funcție care să calculeze produsul scalar a doi vectori. Scrieți un program **Scilab** care să permită introducerea de la tastatură a doi vectori de dimensiune egală și care să afișeze produsul lor scalar.

• Definirea “on-line” a funcțiilor

Funcțiile se pot defini și “on-line” cu comanda **deff**.

Exercițiul F.20: a) Cu ajutorul comenzii **help** aflați ce face comanda **fcontour**.

b) Executați și comentați următoarele comenzi **Scilab**:

```

--> deff('[x] = cerc(u,v)', 'x = sqrt(u^2+v^2)')
--> fcontour(-1:0.1:1,-1:0.1:1,cerc,4)

```

c) Definiți “on-line” funcția “combin” din exercițiul F.18 .

F.7 Alte facilități de postprocesare

Acest exercițiu urmărește exersarea altor facilități de postprocesare ale programului.

Exercițiul F.21: Fie o sarcină punctiformă $q = 10^{-10}$ C, situată în vid ($\epsilon_0 = 8.8 \cdot 10^{-12}$ F/m), în punctul de coordonate $(x, y, z) = (0, 0, 0)$.

Să se vizualizeze, în planul $z = 0$, linii echipotențiale și vectori câmp electric cu ajutorul comenzilor **contour** și **champ**. Pentru aceasta, se vor determina valorile potențialului și ale componentelor câmpului într-o mulțime discretă de puncte din spațiu, plasate în nodurile unei grile generate de un vector de abscise și un vector de ordinate.

Se recomandă parcurgerea următorilor pași:

- Se va scrie o funcție care calculează, într-un punct oarecare, potențialul unei sarcini punctuale situată într-un punct oarecare din spațiu;
- Se va scrie o funcție care calculează, într-un punct oarecare, vectorul componentelor câmpului electric al unei sarcini punctuale, situată într-un punct oarecare din spațiu;
- Se va scrie un program principal, care să permită introducerea datelor problemei de la tastatură și care să afișeze linii echipotențiale și vectori câmp într-un anumit domeniu din planul $z = 0$.

Figurile F.2, F.3, F.4 prezintă rezultatele unui astfel de program în cazul în care grila a fost extrem de rară, generată cu ajutorul vectorilor:

```
d = 0.5
xmin = -d; xmax = d
ymin = -d; ymax = d
pasx = 2*d/3
pasy = 2*d/3
abscise = xmin:pasx:xmax
ordonate = ymin:pasy:ymax
```

Observație: în cazul spectrelor desenate cu comanda **champ**, vectorii sunt desenați (translați) astfel încât mijlocul lor coincide cu punctul de calcul (de aceea, figura F.2 apare oarecum ciudat).

Exercițiul F.22:

a) Observați și comentați alura echipotențialelor pentru diferite grade de finețe ale grilei folosite.

b) Cum tratați cazul în care unul din punctele grilei coincide cu punctul în care se află sarcina?

c) Scrieți un program care să genereze o grilă adaptată problemei, astfel încât liniile echipotențiale să aibă racordări cât mai “dulci”.

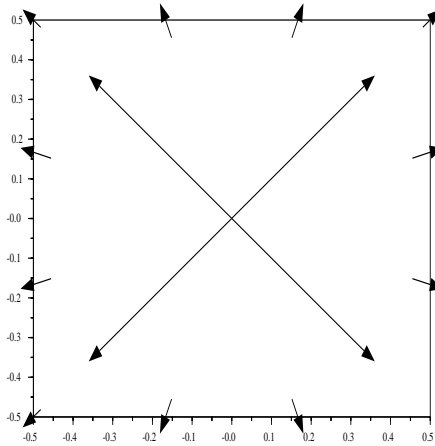


Figura F.2: Efectul comenzii **champ**: vectorii au o lungime proporțională cu valoarea modului lor

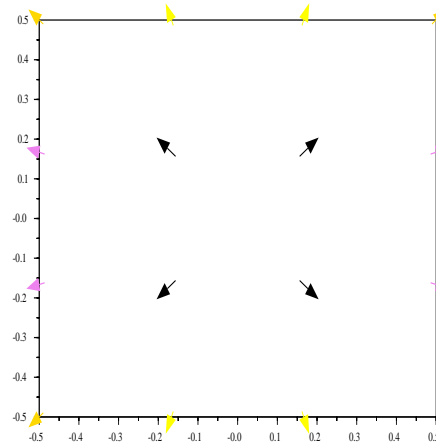


Figura F.3: Efectul comenzii **champ1**: vectorii au lungimi egale și sunt colorați conform valorii modului lor

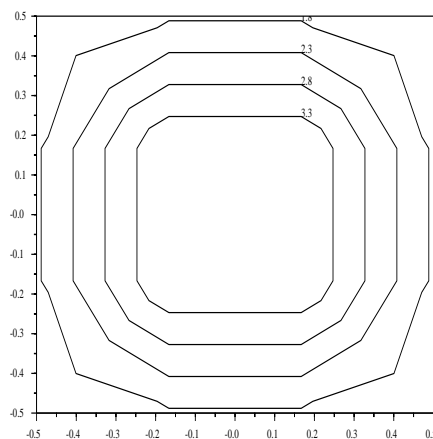


Figura F.4: Efectul comenzii **contour**; curbele au această formă datorită grilei extrem de rare folosite

Bibliografie și webografie

- [1] ***. *AIMMS Home Page*. Paragon Decision Technology, <http://www.paragon.nl/>, 2000.
- [2] ***. *AMPL: A Modeling Language for Mathematical Programming*. Bell Laboratories, <http://www.ampl.com/cm/cs/what/ampl/index.html>, 2000.
- [3] ***. *CUTE - Constrained and Unconstrained Testing Environment*. CLRC Computational Science and Engineering Department, <http://www.cse.clrc.ac.uk/Activity/CUTE>, 2000.
- [4] ***. *GAMS Home Page*. GAMS Development Corp, <http://www.gams.com/solvers/solvers.html>, 2000.
- [5] ***. *MINPACK software*. Netlib Repository at UTK and ORNL, <http://www.netlib.org/minpack>, 2000.
- [6] ***. *NIMBUS - on-line source of optimization service*. University of Jyväskylä, Finland, <http://nimbus.math.jyu.fi/>, 2000.
- [7] ***. *OptiW - Numerische Mathematik Interaktiv*. <http://fb0445.mathematik.tu-darmstadt.de:8081>, 2000.
- [8] ***. *Scilab Home Page*. Institut National de Recherche en Informatique et en automatique, <http://www-rocq.inria.fr/scilab/>, 2000.
- [9] ***. *Testing Electromagnetic Analysis Methods (TEAM) Home Page*. International Compumag Society, <http://ics.ascn3.uakron.edu/>, 2000.
- [10] ***. *The opt directory of Netlib*. <http://www.netlib.org/opt/index.html>, 2000.
- [11] ***. *The Optimization Technology Center*. NEOS server, Argonne National Laboratory, <http://www.ece.nwu.edu/OTC/>, 2000.
- [12] ***. *UniCalc solver*. Russian Research Institute of Artificial Intelligence, <http://www.rriai.org.ru/UniCalc>, 2000.

- [13] ***. *Welcome to the ASCEND Project*. Carnegie Mellon University, <http://www.cs.cmu.edu/~ascend/>, 2000.
- [14] P.R. Adby și M.A.H. Dempster. *Introduction to Optimization Methods*. Chapman and Hall, 1973.
- [15] Bazaraa, Shetty și Sherali. *Nonlinear Programming: Theory and Applications*. Wiley, 1994.
- [16] J.I. Buchanan și P.P. Turner. *Numerical Methods and Analysis*. McGraw-Hill International Editions, 1992.
- [17] G. Ciuprina. *Studiul câmpului electromagnetic în medii neliniare - contribuții privind optimizarea dispozitivelor electromagnetice neliniare*. Teză de doctorat, Universitatea Politehnica București, 1999.
- [18] Coleman și Li. *Large Scale Numerical Optimization*. SIAM Books, 1990.
- [19] Dennis și Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, 1983.
- [20] A. Dolan. *The GA Playground - A general GA toolkit implemented in Java, for experimenting with genetic algorithms and handling optimization problems*. <http://www.aridolan.com/ga/gaa/gaa.html>, 2000.
- [21] R. Fletcher. *Practical Methods of Optimization*. John Wiley & sons, 1987.
- [22] C.A. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*. Kluwer, 1999.
- [23] C.A. Floudas. *The Handbook of Test Problems in Local and Global Optimization*. Princeton University, <http://www.wkap.nl/book.htm/0-7923-5801-5>, 2000.
- [24] R. Fourer. *Nonlinear Programming Frequently Asked Questions*. Optimization Technology Center of Northwestern University and Argonne National Laboratory, <http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html>, 2000.
- [25] R. Fourer. *Software for Optimization: A Buyer's Guide*. AMPL, <http://www.ampl.com/cm/cs/what/ampl/buyers1.html>, 2000.
- [26] Gill, Murray și Wright. *Practical Optimization*. Academic Press, 1981.
- [27] Glover și Laguna. *Tabu Search*. Kluwer, 1997.

-
- [28] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
 - [29] B.S. Gottfried și J. Weisman. *Introduction to Optimization Theory*. Prentice Hall, 1973.
 - [30] R.W. Hamming. *Numerical Methods for Scientists and Engineers*. Dover Publications, Inc., New York, 1973.
 - [31] Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, 1972.
 - [32] Hock și Schittkowski. *Test Examples for Nonlinear Programming Codes*. Springer-Verlag, 1981.
 - [33] Horst, Pardalos și Thoai. *Introduction to global optimization*. Kluwer, 1995.
 - [34] Horst și Tuy. *Global Optimization*. Springer-Verlag, 1993.
 - [35] D.H. Im, S.C. Oark și J.W. Im. Design of Single-Sided Linear Induction Motor Using the Finite Element Method and SUMT. *IEEE Transactions on Magnetics*, vol. 29, nr. 2, pp. 1762–1766, 1993.
 - [36] L. Ingber. *Software for Adaptive Simulated Annealing*. <http://www.ingber.com/ASA-CODE>, 2000.
 - [37] J.S.R. Jang, C.T. Sun și E. Mizutani. *Nero-Fuzzy and Soft Computing*. Prentice Hall, 1997.
 - [38] K. Kadded, R.R. Saldanha și J.L. Coulomb. Mathematical Minimization of the Time Harmonics of the E.M.F. of a DC-PM Machine using a Finite Element Method. *IEEE Transactions on Magnetics*, vol. 29, nr. 2, pp. 1747–1752, 1993.
 - [39] J.L. Kuester și J.H. Mize. *Optimization Techniques with Fortran*. McGraw-Hill Book Company, 1973.
 - [40] A. Kuntsevich. *SolvOpt - Solver for local optimization problems*. <http://bedvgm.kfunigraz.ac.at:8001/alex/solvopt/index.html>, 2000.
 - [41] D.G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, 1984.
 - [42] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag Berlin, 1996.
 - [43] Z. Michalewicz. *Genetic Algorithms/GENOCOP III*. <http://www.coe.uncc.edu/~zbyszek/gchome.html>, 2000.

- [44] H. Mittelmann și P. Spellucci. *Test environment for optimization problems*. <ftp://plato.la.asu.edu/pub/donlp2/testenvirom.tar.gz>, 2000.
- [45] H.D. Mittelmann. *Hans Mittelmann's Benchmarks for Optimization Software*. <http://plato.la.asu.edu/bench.html>, 2000.
- [46] H.D. Mittelmann și P. Spellucci. *Decision Tree for Optimization Software*. <http://plato.la.asu.edu/guide.html>, 2000.
- [47] J.J. More și S. J. Wright. *Optimization Software Guide*, vol. 14 din seria *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, 1993.
- [48] J.J. More și S.J. Wright. *Optimization Software*. <http://www.mcs.anl.gov/otc/Guide/SoftwareGuide>, 2000.
- [49] P. Moscato. *Memetic Algorithms Home Page*. http://www.densis.fee.unicamp.br/~moscato/memetic_home.html, 2000.
- [50] S. Nash și A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, 1996.
- [51] W. Naylor. *WNLIB package*. <http://www.willnaylor.com/wnlib.html>, 2000.
- [52] A. Neumaier. *Global optimization*. <http://solon.cma.univie.ac.at/~neum/glopt.html>, 2000.
- [53] W.H. Press, S.A. Teukolsky, W.T. Vetterling și B.P. Flannery. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, 1992.
- [54] S. Russenschuck. Application of Lagrange Multiplier Estimation to the Design Optimization of Permanent Magnet Synchronous Machines. *IEEE Transactions on Magnetics*, vol. 28, nr. 2, pp. 1525–1528, 1992.
- [55] S. Russenschuck. A Survey of Mathematical Optimization and Inverse Problems in Electromagnetics - with Applications to the design of Superconducting Magnets. *Int. J. of Applied Electromagnetics and Mechanics*, vol. 6, nr. 4, pp. 277–295, 1995.
- [56] R.R. Saldanha, J.L. Coulomb, A. Foggia și J.C. Sabonnadiere. A Dual Method for Constrained Optimization Design in Magnetostatic Problems. *IEEE Transactions on Magnetics*, vol. 27, nr. 5, pp. 4136–4141, 1991.
- [57] R.R. Saldanha, S. Pelissier, K. Kadded adn Y.P. Yonnet și J.L. Coulomb. Nonlinear Optimization Methods Applied to Magnetic Actuators Design. *IEEE Transactions on Magnetics*, vol. 28, nr. 2, pp. 1581–1584, 1992.
- [58] K. Schittkowski. *Nonlinear Programming Codes*. Springer Verlag, 1980.

-
- [59] K. Schittkowski. *Packages for nonlinear optimization*.
<http://www.uni-bayreuth.de/departments/math/org/mathe5/staff/memb/kschittkowski>, 2000.
- [60] N. Takahashi, K. Ebihara, K. Yoshida, T. Nakata, K. Ohashi și K. Miyata. Investigation of Simulated Annealing Method and Its Application to Optimal Design of Die Mold for Orientation of Magnetic Powder. *IEEE Transactions on Magnetics*, vol. 32, nr. 3, pp. 1210–1213, 1996.
- [61] Torn și Zilinskas. *Global Optimization*. Springer-Verlag, 1989.
- [62] C. Udriște și E. Tănăsescu. *Minime și maxime ale funcțiilor reale de variabile reale*. Editura Tehnică, București, 1980.
- [63] D. Whitley. *A Genetic Algorithm Tutorial*. Technical Report CS-93-103, Colorado State University, http://ftp.cs.colostate.edu/pub/TechReports/1993/tr_103.ps.Z, 2000.
- [64] Wismer și Chattergy. *Introduction to Nonlinear Optimization*. North-Holland, 1978.
- [65] X. Yao, Y. Liu și G. Lin. Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation*, vol. 3, nr. 2, pp. 82–102, 1999.
- [66] Zhu și Nocedal. *A limited-memory method*. <ftp://eecs.nwu.edu/pub/lbfgs>, 2000.