STUDIU NUMERIC

COMPARAȚIE ÎNTRE MINOS ŞI HOPDM PENTRU REZOLVAREA PROBLEMELOR DE PROGRAMARE LINIARĂ

Neculai Andrei

nandrei@ici.ro

Institutul Național de Cercetare - Dezvoltare în Informatică, ICI - Bucureşti

Academia Oamenilor de Ştiință din România, Bucureşti

Rezumat: În această lucrare prezentăm câteva rezultate privind performanțele pachetelor MINOS şi HOPDM pentru rezolvarea problemelor de programare liniară. Pachetul MINOS implementează algoritmul simplex cu factorizarea LU a bazei, iar pachetul HOPDM algoritmul de punct interior predictor - corector de ordin superior. În acest sens vom considera un număr de probleme de programare liniară din diverse colecții

Cuvinte cheie: Programarea liniară, MINOS, HOPDM, comparații numerice.

Abstract: In this paper we present the numerical performances of MINOS and HOPDM packages for solving linear programming problems. MINOS implements the simplex algorithm with LU factorization of the basis, and HOPDM uses the predictor-corrector interior-point algorithm. The comparison considers a number of linear programming problems from different collections.

Key words: Linear programming, MINOS, HOPDM, numerical comparisons

1. MINOS

Pachetul MINOS [Murtagh şi Saunders, 1978, 1980, 1982, 1995] reprezintă una dintre cele mai elaborate metode de rezolvare a problemei generale de optimizare neliniară de mari dimensiuni: $\min\{f(x): g(x)=0,\ l\le x\le u\}$, unde $f:\mathbb{R}^n\to\mathbb{R}$, $g:\mathbb{R}^n\to\mathbb{R}^m$ sunt funcții cel puțin de două ori continuu diferențiabile. *În esență acesta implementează o metodă de optimizare secvențială liniară. Ideea metodei MINOS este de a minimiza funcția Lagrange asociată problemei modificată referitor la o aproximație liniară a restricțiilor problemei. MINOS este o extensie a metodei simplex combinată cu o tehnică de gradient redus* [Andrei, 1999, Cap. 25].

Pentru a preciza ideile, în cele ce urmează vom prezenta câteva aspecte esențiale ale metodei MINOS pentru rezolvarea problemei cu *restricții liniare*:

$$\min F(x) = f(x_{\mathrm{N}}) + c^{\mathrm{T}}x \tag{1}$$

referitor la:

$$Ax = b,\quad l\le x\le u,$$

unde A este o $m\times n$- matrice $(m\le n)$ şi vectorul $x\in\mathbb{R}^n$ este partiționat sub forma $x=[x_{\mathrm{N}}\ x_{\mathrm{L}}]^{\mathrm{T}}$, în care x_{N} reprezintă subvectorul variabilelor neliniare, iar x_{L} subvectorul variabilele liniare.

Observăm că A şi c operează asupra tuturor variabilelor. În anumite situații termenul $c^{\mathrm{T}}x_{\mathrm{N}}$ care implică variabilele neliniare este incorporat în $f(x_{\mathrm{N}})$. Presupunem că $f(x_{\mathrm{N}})$ este

cel puțin odată continuu diferențiabilă pe domeniul de admisibilitate al problemei. Chiar dacă partiția lui x şi a lui $F(x)$ în termeni liniari şi neliniari este de o foarte mare importanță practică, totuşi pentru simplificarea prezentării vom nota $F(x)$ şi $\nabla F(x)$ cu $f(x)$, respectiv $g(x)$. Evident, dacă $f(x_{\mathrm{N}}) = 0,$ atunci (8.44) este o problemă de programare liniară.

După cum am văzut, în algoritmul simplex conceptul de soluție admisibilă de bază este fundamental. Virtutea acestui concept constă în modul în care tratează restricțiile margini simple asupra variabilelor: $l \le x \le u$. Într-adevăr, atât din punct de vedere teoretic cât şi practic nu este recomandabil ca restricțiile margini simple să fie incluse în matricea A. Mult mai bine este ca acestea să fie utilizate pentru eliminarea variabilelor. Algoritmul simplex cu variabile mărginite realizează într-un mod foarte eficient acest lucru, operând numai asupra bazei B şi nu asupra întregii matrice A.

În cazul problemelor neliniare nu ne putem aştepta ca o soluție optimă să fie soluție admisibilă de bază ca în algoritmul simplex. Totuşi, dacă numărul variabilelor neliniare este mic, atunci pare rezonabil să presupunem că o soluție optimă este „*aproape*" o soluție admisibilă de bază. Astfel, ca o generalizare, Murtagh şi Saunders [1978] au introdus conceptul de „*variabile superbazice*". Partiționând vectorul x sub forma $x = [x^{\mathrm{B}}\ x^{\mathrm{S}}\ x^{\mathrm{N}}]^{\mathrm{T}}$, atunci restricțiile din (1) devin:

$$Ax = Bx^{\mathrm{B}} + Sx^{\mathrm{S}} + Nx^{\mathrm{N}} = b. \tag{2}$$

Matricea B este $m \times m$ - dimensională şi nesingulară. Matricea S este $m \times s$ - dimensională cu $0 \le s \le n - m,$ iar N conține restul coloanelor din A. Variabilele x^{B}, x^{S} şi x^{N} sunt numite: *variabile bazice* (de bază), *superbazice* şi respectiv *nebazice*. Atât variabilele bazice cât şi cele superbazice sunt libere de a lua valori între marginile lor inferioare respectiv superioare. Variabilele superbazice sunt „*variabile de forțare*" în sensul că ele pot fi deplasate în orice direcție cu scopul de a îmbunătăți valoarea funcției obiectiv. Ca în metoda simplex, variabilele nebazice sunt fixate la una din marginile lor, inferioară sau superioară. Variabilele bazice sunt obligate să ia valori într-un mod bine definit pentru a menține admisibilitatea în raport cu restricțiile $Ax = b$.

Presupunem că $f(x)$ se poate dezvolta în serie Taylor sub forma:

$$f(x + \Delta x) = f(x) + g(x)^{\mathrm{T}} \Delta x + \frac{1}{2} \Delta x^{\mathrm{T}} H(x + \gamma \Delta x) \Delta x, \tag{3}$$

unde $0 \le \gamma \le 1$ este un parametru şi $H(x + \gamma \Delta x)$ este hesseianul lui f calculat într-un punct de pe segmentul de linie cu capetele în $x,$ respectiv $x + \Delta x$. Observăm că dacă f este pătratică, atunci H este o matrice constantă. Dacă dispunem de un punct curent admisibil $x,$ problema este de a construi o deplasare Δx astfel încât următoarele două proprietăți să fie îndeplinite:

Proprietatea 1 (Admisibilitate):

$$\begin{bmatrix} B & S & N \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \Delta x^{\mathrm{B}} \\ \Delta x^{\mathrm{S}} \\ \Delta x^{\mathrm{N}} \end{bmatrix} = 0. \tag{4}$$

Proprietatea 2 (Optimalitate):

$$\begin{bmatrix} g_{\mathrm{B}} \\ g_{\mathrm{S}} \\ g_{\mathrm{N}} \end{bmatrix} + H \begin{bmatrix} \Delta x^{\mathrm{B}} \\ \Delta x^{\mathrm{S}} \\ \Delta x^{\mathrm{N}} \end{bmatrix} = \begin{bmatrix} B^{\mathrm{T}} & 0 \\ S^{\mathrm{T}} & 0 \\ N^{\mathrm{T}} & I \end{bmatrix} \begin{bmatrix} \mu \\ \lambda \end{bmatrix}. \tag{5}$$

Dacă $x + \Delta x$ este suficient de aproape de x, atunci se poate utiliza un „model pătratic" al funcției f în jurul lui x, astfel încât membrul stâng din (5) reprezintă gradientul funcției obiectiv în $x + \Delta x$. Pentru ca $x + \Delta x$ să fie un punct de optim local în raport cu mulțimea curentă a restricțiilor active vom impune ca gradientul funcției obiectiv să fie ortogonal la suprafața formată din mulțimea restricțiilor active. Pentru aceasta, gradientul trebuie să fie o combinație liniară a normalelor restricțiilor active, dată de membrul drept din (5) [Andrei, 1999].

Parametrii μ şi λ sunt multiplicatorii Lagrange. Mai mult, pentru optimalitatea lui $x + \Delta x$ trebuie ca gradientul cu semn schimbat să fie direcționat ortogonal în afara domeniului de admisibilitate definit de restricțiile problemei. Pentru (1) vom impune deci ca $\lambda_j \le 0$ dacă $x_j^{\mathrm{N}} = u_j$, sau $\lambda_j \ge 0$ dacă $x_j^{\mathrm{N}} = l_j$, $j = 1, \ldots, n - m - s$.

Din (4) obținem imediat:

$$\begin{aligned} B\Delta x^{\mathrm{B}} + S\Delta x^{\mathrm{S}} &= 0, \\ \Delta x^{\mathrm{N}} &= 0. \end{aligned} \tag{6}$$

Deci

$$\Delta x^{B} = -W\Delta x^{\mathrm{S}}, \tag{7}$$

unde

$$W = B^{-1}S. \tag{8}$$

Ca atare,

$$\Delta x = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta s^{\mathrm{S}}, \tag{9}$$

ceea ce înseamnă că putem lucra numai cu vectorul Δx^{S}, care este de dimensiune s, mult mai mică decât n.

Matricea W din (8) nu se calculează niciodată, deoarece B^{-1} este reprezentată sub forma de eliminare a inversei. Aceasta este esența algoritmului MINOS. Observăm că MINOS este un algoritm de tip simplex în care operațiile esențiale cu B^{-1} privind: calculul factorilor de cost redus, actualizarea coloanei pivot şi actualizarea soluției, sunt realizate utilizând forma de eliminare a inversei (factorizarea LU) a bazei pe care am detaliat-o în acest capitol. Este foarte clar că algoritmul MINOS conține o serie întreagă de ingrediente proprii rezolvării problemelor de optimizare cu restricții neliniare. Totuşi, importantă aici este forma de reprezentare a inversei bazei sub forma factorizată LU şi utilizarea ei în toate operațiile care o implică în cadrul algoritmului simplex.

Vom considera un număr de 93 de probleme de programare liniară din colecţiile *Netlib* [Gay, 1985] şi *Kennington* [Carolan, Hill, Kennington, Niemi şi Wichmann, 1990], pe care le vom rezolva cu algoritmul MINOS.

Tabelul 1 conţine caracteristicile problemelor considerate (m - numărul de restricţii, n - numărul de variabile, nz - numărul de elemente nenule în matricea problemei) împreună cu rezultatele optimizării (*iter* - numărul de iteraţii, z - valoarea optimă a funcţiei obiectiv găsită de MINOS, *cpu* - timpul de calcul - secunde).

Tabelul 1. Rezultate MINOS.
***cpu* - secunde.**

Nume Prob.	Caracteristicile problemei			Optimizare		
	m	*n*	*nz*	*iter*	*z*	*cpu*
0	1	2	3	4	5	6
25fv47	821	1571	11127	7760	0.5501845e4	3.93
adlittle	56	97	465	100	0.2254949e6	0.03
afiro	27	32	88	9	-0.464753e3	0
agg	488	163	2541	115	-0.359917e8	0.05
agg2	516	302	4515	156	-0.202392e8	0.07
agg3	516	302	4531	169	0.1031211e8	0.05
band	125	129	754	7	0.7718397e3	0
bandm	305	472	2659	476	-0.158628e3	0.11
beaconf	173	262	3476	45	0.3359248e5	0.01
blend	74	83	521	78	-0.308121e2	0
bnl1	643	1175	6129	1429	0.1977629e4	0.42
bnl2	2324	3489	16124	4652	0.1811236e4	4.23
boeing1	351	384	3865	530	-0.335213e3	0.11
boeing2	166	143	1339	183	-0.315018e3	0.01
brandy	220	249	2150	321	0.1518509e4	0.05
capri	217	353	1786	262	0.2690012e4	0.05
cetsud	1906	3347	7262	2602	0.7839141e9	1.43
ch	3852	5062	42910	26226	0.9257563e6	39.56
co5	5878	7993	92788	30938	0.7144723e6	81.28
cq5	5149	7530	83564	37280	0.4001338e6	85.06
cre-a	3516	4067	19054	4283	0.2359540e8	4.89
cre-c	3068	3678	16922	4684	0.2527511e8	4.70
czprob	929	3523	14173	1587	0.2185196e7	0.59
cycle	1903	2857	21322	2678	-0.522639e1	2.08
d2q06c	2171	5167	35081	44974	0.1227842e6	53.47
d6cube	415	6184	43888	106846	0.3154916e3	44.81
degen2	444	534	4449	997	-0.143517e4	0.25
degen3	1503	1818	26230	6881	-0.987293e3	6.10
e226	223	282	2767	459	-0.116389e2	0.06

Tabelul 1. Rezultate MINOS. (continuare)
cpu **- secunde.**

0	1	2	3	4	5	6
etamacr	400	688	2489	624	-0.755715e3	0.13
ffff800	524	854	6235	477	0.555679e6	0.14
finnis	497	614	2714	486	0.1727910e6	0.11
fit1d	24	1026	14430	2411	-0.914637e4	0.19
fit1p	627	1677	10894	843	0.9146378e4	0.44
forplan	161	421	4916	323	-0.664218e3	0.05
ganges	1309	1681	7021	689	-0.109585e6	0.31
grenbea	2392	5405	31499	22826	-0.725552e8	24.01
grenbeb	2392	5405	31499	12455	-0.430223e7	13.22
grow7	140	301	2633	108	-0.477878e8	0.03
grow15	300	645	5665	291	-0.106870e9	0.10
grow22	440	946	8318	422	-0.160834e9	0.19
israel	174	142	2358	281	-0.896644e6	0.03
kb2	43	41	291	51	-0.174990e4	0
ken-07	2426	3602	11981	1826	-0.679520e9	1.45
lotfi	153	308	1086	200	-0.252647e2	0.01
maros	846	1443	10006	2427	-0.580637e5	1.02
marosr7	3136	9408	151120	2520	0.1497185e7	8.36
nesm	663	2923	13988	3435	0.1407607e8	1.05
optanyrf	528	1077	8830	2330	-0.450173e6	0.70
p05	5090	9500	68455	1675	0.5560002e6	3.82
p2756	755	2756	11103	818	0.2688750e4	0.25
pcb1000	1565	2428	22499	3216	0.5680945e5	2.43
pcb3000	3961	6810	63367	9971	0.1374164e6	17.70
pds-02	2953	7535	21252	2940	0.288578e11	2.78
perold	625	1376	6026	3799	-0.938075e4	1.47
pilot	1441	3652	43220	16273	-0.557412e3	30.07
pilot4	410	1000	5145	1314	-0.258113e4	0.40
pilotja	940	1988	14706	6514	-0.611307e4	3.41
pilotwe	722	2789	9218	5303	-0.272010e7	2.00
qap8	912	1632	8304	9463	0.203500e3	9.17
qap12	3192	8856	44244	258737	0.522894e3	1960.75
r05	5190	9500	113455	1712	0.5578318e6	5.37
recipe	91	180	752	27	-0.266616e3	0
refine	30	33	155	26	-0.392691e6	0
sc105	105	103	281	27	-0.522020e2	0
sc205	205	203	552	52	-0.522020e2	0
scfxm1	330	457	2612	384	0.1841675e5	0.08
scfxm2	660	914	5229	719	0.3666026e5	0.22
scfxm3	990	1371	7846	1092	0.5490125e5	0.47
scorpio	388	358	1708	155	0.1878124e4	0.03
seba	515	1028	4874	306	0.1571160e5	0.08
shell	536	1775	4900	338	0.120882e10	0.09
ship04l	402	2118	8450	273	0.1793324e7	0.09
ship04s	402	1458	5810	169	0.1798714e7	0.07

Tabelul 1. Rezultate MINOS. (continuare)
***cpu* - secunde.**

0	1	2	3	4	5	6
ship08l	778	4283	17085	438	0.1909055e7	0.24
ship08s	778	2387	9501	256	0.1920098e7	0.14
sierra	1227	2036	9252	1084	0.1539443e8	0.47
stair	356	467	3857	512	-0.251266e3	0.17
stocfor1	117	111	474	56	-0.411319e5	0.02
stocfor2	2157	2031	9492	1876	-0.390244e5	1.39
tuff	333	587	4523	434	0.2921477e0	0.08
vtpbase	198	203	914	163	0.1298314e6	0.02
woodw	1098	8405	37478	3818	0.1304476e1	2.63

Tabelul 2 conţine rezultatele furnizate de MINOS pentru rezolvarea unor probleme de dimensiuni ceva mai mari [Suhl, 2010].

Tabelul 2. Rezultate MINOS
Probleme de mari dimensiuni.
***cpu* - secunde.**

Nume Prob.	Caracteristicile problemei			Optimizare		
	m	*n*	*nz*	*iter*	*z*	*cpu*
0	1	2	3	4	5	6
cq9	9451	13778	157598	104150	0.5055445e6	449.47
cre-b	9649	72447	328542	183473	0.2312964e8	678.23
cre-d	8927	69980	312626	240460	0.2445497e8	829.64
ge	10339	11098	53763	15524	0.5581281e7	51.39
ken-11	14694	21349	70354	15307	-0.69723e10	71.21
ken-13	28633	42659	139834	45461	-0.10257e10	440.56
pds-10	16559	48763	140063	74814	0.267270e11	404.45
stocfor3	16675	15695	74004	14082	-0.399767e5	72.55
osa-07	1118	23949	167643	1915	0.5357225e6	2.02
osa-14	2338	52460	367220	4116	0.1106462e7	7.91

Câteva comentarii sunt necesare.

1. Experimentele numerice efectuate în acest studiu au fost executate pe un PC Intel Pentium V, frecvenţa 3.40GHz, 2GB memorie RAM.
2. Algoritmul simplex în implementarea MINOS (versiunea 5.1) este capabil să rezolve o mare varietate de probleme de programare liniară, de diverse dimensiuni şi structuri. Pentru probleme de mici dimensiuni, de până la 1000 de restricţii, timpul de calcul este aproape nesesizabil de sistemul de calcul utilizat (sub o secundă). Aceasta nu este o regulă, deoarece timpul de calcul depinde într-o manieră cvasi-necunoscută de foarte multe caracteristici ale problemei incluzând aici: numărul de variabile, numărul de restricţii, numărul de elemente nenule ale matricei, structura zero/nezero (tiparul) a matricei, mărimea elementelor nenule ale problemei, etc.
3. Observăm că nu se poate stabili o relaţie între numărul de iteraţii şi timpul de calcul necesare rezolvării unei probleme de programare liniară cu algoritmul MINOS. Se pare că importantă este structura matricei problemei, care impune o anumită sparsitate factorilor L şi U din factorizarea LU a bazei. În general, cu cât numărul de iteraţii este mai mare cu atât timpul de calcul este mai lung, dar dependenţa este foarte neliniară şi nepredictibilă. De exemplu, pentru problema „d6cube" algoritmul MINOS necesită 106846 iteraţii şi 44.81 secunde pentru a găsi soluţia optimă. Dar, pentru „qap12" sunt necesare 258737

(=106846 × 2.42) iteraţii şi 1960.75 (=44.81 × 43.76) secunde. În capitolul 10 prezentăm complexitatea algoritmului simplex, precum şi anumite consideraţii empirice privind numărul de iteraţii ca o funcţie de numărul de restricţii. Studii numerice intensive arată că numărul de iteraţii de obicei este cuprins între m şi $4m$, când $n \cong 3m$. Numai rareori acest număr depăşeşte $10m$. Din exemplele de mai sus vedem precaritatea conjecturilor în acest domeniu.

4. În tabelul 2 prezentăm rezultatele furnizate de algoritmul MINOS pentru rezolvarea unor probleme de programare liniară de mari dimensiuni. Vedem că cea mai mare este problema „ken-13", care are 28633 de restricţii, 42659 variabile şi 139834 elemente nenule în matricea coeficienţilor. Problema a fost rezolvată în 7.34 minute. În aceeaşi clasă se poate plasa şi problema „pds-10" cu 16559 restricţii, 48763 variabile şi 140063 elemente nenule, care a fost rezolvată în 6.74 minute. Şi în acest caz al problemelor de dimensiuni mai mari se vede o dispersie foarte mare, total nepredictibilă, a numărului de iteraţii şi a timpului de calcul în funcţie de dimensiunile problemei. Totuşi, important este faptul că algoritmul simplex cu factorizarea LU a bazei poate rezolva probleme de programare liniară de mari dimensiuni pe calculatoare personale foarte comune.
5. Ca o concluzie a acestui studiu numeric putem afirma că algoritmul simplex (aici în implementarea MINOS) este un algoritm de încredere, care este capabil să rezolve probleme de programare liniară, modele ale unor fenomene fizice reale.

2. HOPDM

Pachetul HOPDM, elaborat de Jacek Gondzio [1993, 1996], utilizează *metoda primal – dual de punct interior bazată pe funcţia barieră logaritmică cu aproximarea de ordinul doi a traiectoriei* ($m_k = 2$). În factorizarea Cholesky se utilizează permutarea dată de *algoritmul gradului minim*. Înaintea rezolvării problemei, pachetul HOPDM execută o preprocesare a matricei, inclusiv scalarea acesteia.

Vom considera un număr de 86 de probleme de programare liniară din colecţiile *Netlib* [Gay, 1985] şi *Kennington* [Carolan, Hill, Kennington, Niemi şi Wichmann, 1990], precum şi alte probleme de programare liniară generate pentru acest studiu numeric.

Tabelul 3 conţine rezultatele optimizării cu HOPDM pentru problemele de dimensiuni reduse incluzând: *caracteristicile problemelor considerate* (m - numărul de restricţii, n - numărul de variabile, nz - numărul de elemente nenule în matricea problemei) împreună cu *rezultatele algoritmului de permutare de grad minim* (AA^{T} - numărul elementelor subdiagonale din matricea AA^{T}, L - numărul elementelor subdiagonale din factorul Cholesky al matricei AA^{T}), precum şi *rezultatele optimizării* ($iter$ - numărul de iteraţii, z^* - valoarea optimă a funcţiei obiectiv găsită de HOPDM, cpu - timpul de calcul în secunde).

Tabelul 3. Rezultate HOPDM.
***cpu* secunde.**

Nume	Caracteristicile problemei			Factorizare		Optimizare		
Prob.	m	n	nz	AA^{T}	L	$iter$	z^*	cpu
0	1	2	3	4	5	6	7	8
25fv47	821	1571	11127	10456	32238	24	0.5501845e4	1.39
adlittle	56	97	465	308	340	11	0.2254949e6	0.07
afiro	27	32	88	57	74	7	-0.464753e3	0.48
agg	488	163	2541	6953	9215	17	-0.359917e8	0.67
agg2	516	302	4515	11143	19322	16	-0.202392e8	0.68

agg3	516	302	4531	11153	19322	16	0.1031211e8	1.04
band	125	129	754	490	1000	13	0.7718397e3	0.39
bandm	305	472	2659	2121	3368	17	-0.158628e3	1.01
beaconf	173	262	3476	471	482	12	0.3359248e5	0.18
blend	74	83	521	526	726	9	-0.308121e2	0.70
bnl1	643	1175	6129	3973	10317	24	0.1977629e4	0.39
bnl2	2324	3489	16124	12242	81245	21	0.1811236e4	2.48
boeing1	351	384	3865	3710	5298	19	-0.335213e3	0.54
boeing2	166	143	1339	1151	1864	15	-0.315018e3	1.04
brandy	220	249	2150	1662	2223	19	0.1518509e4	0.90
capri	271	353	1786	1835	3486	17	0.2690012e4	1.00
cetsud	1906	3347	7262	2663	4034	17	0.7839141e9	0.29
ch	3852	5062	42910	20347	91896	23	0.9257563e6	3.29
co5	5878	7993	92788	48736	147980	44	0.7144695e6	7.23
cq5	5149	7530	83564	44347	135757	33	0.4001338e6	5.51
cre-a	3516	4067	19054	17433	29663	27	0.2359540e8	1.39
cre-c	3068	3678	16922	14615	26439	26	0.2527511e8	1.17
czprob	929	3523	14173	2688	2790	26	0.2185196e7	0.75
cycle	1903	2857	21322	18376	54809	30	-0.522639e1	2.46
d2q06c	2171	5167	35081	24860	127569	28	0.1227842e6	6.06
d6cube	415	6184	43888	13036	54470	17	0.3154916e3	2.35
degen2	444	534	4449	6865	15365	11	-0.143517e4	0.21
degen3	1503	1818	26230	49759	119202	15	-0.987293e3	2.95
e226	223	282	2767	2163	2992	18	-0.187519e2	0.45
etamacr	400	688	2489	2124	9895	20	-0.755715e3	1.04
ffff800	524	854	6235	5309	8251	26	0.555679e6	1.15
finnis	497	614	2714	1854	3679	19	0.1727910e6	0.45
fit1d	24	1026	14430	267	272	21	-0.914637e4	0.43
fit1p	627	1677	10894	99016	152453	16	0.9146378e4	5.68
forplan	161	421	4916	1819	2659	22	-0.664218e3	0.75
ganges	1309	1681	7021	6627	11384	19	-0.109585e6	1.03
grenbea	2392	5405	31499	26644	46894	64	-0.725552e8	3.75
grenbeb	2392	5405	31499	26532	45167	33	-0.430226e7	2.06
grow7	140	301	2633	1450	3090	12	-0.477878e8	0.90
grow15	300	645	5665	3130	7090	15	-0.106870e9	0.60
grow22	440	946	8318	4600	10552	16	-0.160834e9	1.06
israel	174	142	2358	7010	8536	18	-0.896644e6	0.21

Tabelul 3. Rezultate HOPDM. (continuare)
cpu **secunde.**

0	1	2	3	4	5	6	7	8
kb2	43	41	291	357	421	13	-0.174990e4	0.53
ken-07	2426	3602	11981	3136	5797	13	-0.679520e9	0.50
lotfi	153	308	1086	588	977	11	-0.252647e2	0.48
maros	846	1443	10006	6762	11935	23	-0.580637e5	0.34
marosr7	3136	9408	151120	161040	519612	10	0.1497185e7	14.42
nesm	663	2923	13988	4057	20407	24	0.1407603e8	1.17
p05	5090	9500	68455	100884	233965	26	0.5560002e6	7.26
p2756	755	2756	11103	5490	5716	13	0.2688750e4	1.26
pcb1000	1565	2428	22499	13754	28300	20	0.5680945e5	1.45
pcb3000	3961	6810	63367	35953	100340	22	0.1374164e6	3.48

pds-02	2953	7535	21252	9515	40289	19	0.288578e11	1.54
perold	625	1376	6026	5454	22410	38	-0.938075e4	1.29
pilot	1441	3652	43220	58265	191454	23	-0.557489e3	9.40
pilot4	410	1000	5145	5753	14409	37	-0.258113e4	0.95
pilotja	940	1988	14706	11188	39993	33	-0.611313e4	1.82
pilotwe	722	2789	9218	4464	14714	43	-0.272010e7	1.54
qap8	912	1632	8304	13952	193032	7	0.203500e3	3.29
qap12	3192	8856	44244	74592	2135388	15	0.522894e3	286.70
r05	5190	9500	113455	185699	449030	21	0.5578318e6	17.37
recipe	91	180	752	135	144	7	-0.266616e3	0.18
sc105	105	103	281	226	595	8	-0.522020e2	0.37
refine	30	33	155	64	75	9	-0.392691e6	0.79
sc205	205	203	552	451	1287	9	-0.522020e2	0.31
scfxm1	330	457	2612	2372	4013	20	0.1841675e5	0.51
scfxm2	660	914	5229	4763	8083	22	0.3666026e5	1.04
scfxm3	990	1371	7846	7154	12129	22	0.5490125e5	1.17
seba	515	1028	4874	1	1	6	0.1571160e5	0.31
shell	536	1775	4900	1406	4006	21	0.120882e10	0.93
ship04l	402	2118	8450	2320	2392	13	0.1793324e7	0.32
ship04s	402	1458	5810	1533	1605	11	0.1798714e7	0.78
ship08l	778	4283	17085	3864	4025	14	0.1909055e7	1.00
ship08s	778	2387	9501	1911	2057	13	0.1920098e7	0.75
sierra	1227	2036	9252	4418	12555	18	0.1539436e8	0.71
stair	356	467	3857	6197	14417	16	-0.251266e3	0.35
stocfor1	117	111	474	374	706	9	-0.411319e5	0.6
stocfor2	2157	2031	9492	9806	27418	18	-0.390244e5	0.48
tuff	333	587	4523	3796	6219	17	0.2921477e0	0.82
vtpbase	198	203	914	222	291	9	0.1298314e6	0.62
woodw	1098	8405	37478	12611	29997	26	0.1304476e1	1.39

Tabelul 4 conţine rezultatele şi performanţele pachetului HOPDM pentru probleme de programare liniară de mari dimensiuni, împreună cu caracteristicile acestor probleme [Suhl, 2010].

Tabelul 4. Rezultate HOPDM.
Probleme de mari dimensiuni.
cpu **secunde.**

Nume	Caracteristicile problemei			Factorizare		Optimizare		
Prob.	*m*	*n*	*nz*	AA^T	*L*	*iter*	z^*	*cpu*
0	1	2	3	4	5	6	7	8
cq9	9451	13778	157598	84366	361104	38	0.5055417e6	20.37
ge	10339	11098	53763	44832	263443	35	0.5581273e7	12.29
ken-11	14694	21349	70354	20812	62468	21	-0.69723e10	2.96
stocfor3	16675	15695	74004	80640	205791	32	-0.399767e5	5.73
osa-07	1118	23949	167643	2343	2345	9	0.535722e6	1.95

Câteva comentarii sunt necesare.

1. Şi în acest studiu numeric experimentele a fost efectuate pe un PC Intel Pentium V, frecvenţa 3.40GHz, 2GB memorie RAM.

2. Observăm cum algoritmul de permutare de grad minim calculează umplerea în factorul L din factorizarea matricei simetrice AA^T (vezi coloanele 4 şi 5 din tabelele 3 şi 4. Umplerea depinde de structura matricei A. Aceasta are un impact foarte pronunțat asupra performanțelor algoritmului HOPDM.

3. Remarcăm imediat că, independent de dimensiunea problemelor rezolvate, algoritmul de punct interior (în implementarea HOPDM) execută sub 50 de iterații. Pentru probleme de mici dimensiuni, algoritmul simplex (în implementarea MINOS) este mai rapid. Totuşi, pentru probleme de mari dimensiuni, algoritmii de punct interior tind să fie superiori şi ca număr de iterații şi ca timp de calcul.

4. Este foarte clar că diferențele dintre cele două moduri de abordare sunt profunde. Algoritmul simplex parcurge un drum, din vârf în vârf, de-a lungul muchiilor restricțiilor, având un *caracter miop local*, nefăcând nicio încercare de a merge pe fețele sau prin interiorul poliedrului restricțiilor. Pe de altă parte, algoritmii de punct interior *deplasează iterațiile prin interiorul domeniului de admisibilitate de-a lungul traiectoriei centrale, acomodându-se mai bine, cel puțin din punctul de vedere al iterațiilor, structurii poliedrului restricțiilor*.

5. Totuşi, lucrurile nu sunt ce par a fi. Vedem că algoritmul HOPDM nu poate rezolva câteva probleme de mari dimensiuni din acest studiu numeric (cre-b, cre-d, ken-13, pds-10, osa-14) pe care algoritmul MINOS le poate rezolva. Dificultatea întâmpinată de HOPDM în ceea ce priveşte rezolvarea acestor probleme constă în faptul că umplerea în factorul L este prea mare şi creează dificultăți numerice algoritmului de factorizare. Pe de altă parte, algoritmul simplex în implementarea MINOS nu se confruntă cu asemenea probleme deoarece la fiecare iterație operează cu o inversă a bazei sub formă de produs, care implică un număr mult mai mic de elemente nenule. De exemplu, pentru problema „ken-11" algoritmul HOPDM determină o umplere în factorul Cholesky L de 62468 elemente nenule, pe când algoritmul MINOS generează numai 48089 elemente nenule în factorizarea LU a bazei. În aceeaşi ordine de idei, menționăm că pentru cea mai mare problemă rezolvată de MINOS, problema PDS-10, umplerea în factorizarea LU este 6766 elemente nenule în factorul L şi de 35843 de elemente nenule în factorul U, care conduce la un total de 42609 elemente nenule.

3. Comparație MINOS versus HOPDM

Rezultatele obținute cu HOPDM sunt comparate cu cele obținute cu algoritmul simplex cu forma de eliminare a inversei în implementarea MINOS.

Dacă definim performanța relativă la timpul de calcul pentru doi algoritmi A şi B ca $time^i_{A-B} = -\log_2(time^i_A / time^i_B)$, unde $time^i_A$ reprezintă timpul cpu corespunzător algoritmului A pentru a rezolva a i-a problemă, atunci performanța relativă la timpul de calcul lui HOPDM versus MINOS, pentru acest studiu numeric de 86 de probleme rezolvate de ambii algoritmi, este arătată în figura 1.

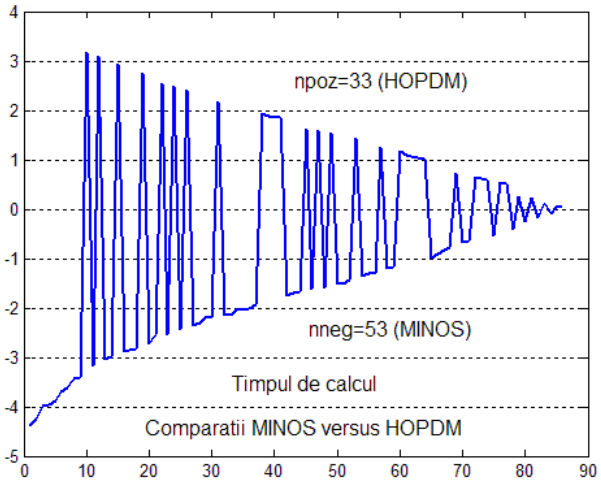

Figura 1. Performanța relativă la timpul de calcul HOPDM-MINOS.

Semnul lui $time^i_{A-B}$ indică algoritmul câştigător. Dacă $time^i_{A-B}$ este pozitiv, atunci algoritmul A este superior din punctul de vedere al timpului de calcul pentru rezolvarea problemei i, ($time^i_A < time^i_B$). Vedem că algoritmul MINOS este uşor superior din acest punct de vedere. Ca atare, algoritmul simplex rămâne ca algoritmul de bază pentru rezolvarea problemelor de programare liniară.

4. Concluzii

Concluzionăm acest studiu numeric cu ideea că ambele moduri de abordare (metoda simplex şi metodele de punct interior de urmărire a traiectoriei) sunt operaționale pentru rezolvarea problemelor de programare liniară, constituind un arsenal redutabil bine fundamentat atât din punctul de vedere al convergenței şi complexității computaționale cât şi al implementării în programe de calcul [Andrei, 2011].

BIBLIOGRAFIE

1. **ANDREI, N.**: Programarea matematică avansată. Teorie, metode computaționale, aplicații. Editura Tehnică, Bucureşti, 1999.

2. **ANDREI, N.**: Critica Rațiunii Algoritmilor de Programare Liniară. Editura Academiei Române, Bucureşti. (în curs de apariție).

3. **CAROLAN, W. J., J. E. HILL, J. L. KENNINGTON, S. NIEMI, S. J. WICHMANN**: An empirical evaluation of the KORBX algorithms for military airlift applications. Operations Research, vol.38, no.2, March-April 1990, pp. 240-248.

4. **GAY, D. M.**: Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Newsletter, 1985.

5. **GONDZIO, J.**: Implementing Cholesky factorization for interior point methods of linear programming. Optimization, vol.27, 1993, pp. 121-140.

6. **GONDZIO, J.**: Multiple centrality corrections in a primal-dual method for linear programming. Computational Optimization and Applications, vol.6, no.2, September 1996, pp. 137-156.

7. **MURTAGH, B. A., M. A. SAUNDERS**: Large-scale linearly constrained optimization. Mathematical Programming, vol.14, 1978, pp. 41-72.

8. **MURTAGH, B. A., M. A. SAUNDERS**: MINOS/AUGMENTED user's manual. Technical Report SOL 80-14, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, CA 94305, 1980.

9. **MURTAGH, B. A., M. A. SAUNDERS**: A projected lagrangian algorithm and its implementation for sparse nonlinear constraints. Mathematical Programming Study, vol.16, 1982, pp. 84-117.

10. **MURTAGH, B. A., M. A. SAUNDERS**: MINOS 5.4 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, CA 94305, February 1995.

11. **SUHL, W.**: Large-scale linear programming problems. Freie Universität Berlin, Private communication on large-scale linear programming problems. May 28, 2010.