CODEN:LUTFD2/(TFRT-5294)/0-061/(1983)

ELLIPSOIDMETODEN: KHACHIJANS ALGORITM FÖR LINJÄR PROGRAMMERING

RONNY TOCAJ

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA

APRIL 1983

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Report
Date of issue
April 1983
Document number
CODEN:LUTFD2/(TFRT-5294)/0-061/(1983)

Author(s)
Ronny Tocaj

Supervisor
Per Olof Gutman

Sponsoring organization

Title and subtitle
Ellipsoidmetoden: Khachijans Algoritm för Linjär Programmering.
(The Ellipsoid method:Khachijans Algorithm for Linear Programming)

Abstract
This thesis treats the ellipsoid method for linear programming. The method is described both from a theoretical and a practical point of view. A subroutine, LPEM, which solves LP-problems with the ellipsoid method, is presented and described. LPEM is written in Pascal. Finally some computational results from running LPEM on two testexampels, Klee-Minty's problem and the double integrator controlled by the LP-regulator, are presented and discussed.

Key words
The ellipsoid method, Khachihan's algorithm, Linear programming.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
61

Recipient's notes

Security classification

FÖRORD.

År 1979 besvarades en viktig teoretisk fråga inom området linjär programmering. Svaret gavs av den ryska matematikern Khachijan i artikeln "A polynomial algorithm in linear programming" införd i Doklady Akademiia Nauk SSSR. Han visade nämligen att antalet iterationer som krävs för att lösa linjärprogrammeringsproblem är högst polynomiellt i dimensionen av problemet om ellipsoidmetoden används (ellipsoidmetoden utvecklades ursprungligen av Shor för att användas inom området konvex programmering). Resultatet väckte stort uppseende världen över i datalogikretsar och figurerade faktiskt också som förstasidestoff i dagspressen.

I den här uppsatsen ges först en tämligen omfattande teoretisk beskrivning av ellipsoidmetoden. Därefter presenteras ett program som löser linjärprogrammerings-problem med metoden. Slutligen redovisas resultat från datorkörningar med detta program på två testexempel.

Jag vill här passa på att tacka mina handledare dr. Per-Olof Gutman och prof. Lars Gårding för deras hjälp under arbetets gång. Framför allt är jag tacksam för att jag har fått fria händer att utforma mitt arbete, även om resultaten därigenom kanske blivit klenare än vad som annars hade varit fallet. Ett speciellt tack vill jag rikta till prof. Gårding för att han presenterat egna LP-algoritmer; faktiskt har arbetet med dessa varit det mest stimulerande trots att tyvärr inget av det redovisas i denna uppsats.

Ronny Tocaj

INNEHALL.

1. TEORETISK BESKRIVNING AV ELLIPSOIDMETODEN.

I det här kapitlet ges en teoretisk beskrivning av ellipsoidmetoden. Iterationsformlerna härleds och begrepp som djupa snitt, parallella snitt och kombinationssnitt tas upp. Dessutom beskrivs några olika sätt att lösa linjärprogrammeringsproblem med metoden, samt hur metoden kan implementeras. Innehållet i kapitlet bygger främst på referenserna [2], [4] och [6].

1.1 ITERATIONSFORMLERNA I ELLIPSOIDMETODEN.

Principen i ellipsoidmetoden är följande: givet ett elliptiskt klot E och ett halvrum H så kan man konstruera ett elliptiskt klot E^* med minsta volym som innehåller snittmängden E∩H, se fig (1.1).

Detta kan användas för att lösa LP-problem, dvs för att maximera en linjär form $c^t x$, $x \in R^n$, över en polyeder

$$Q: \quad a_i^t x \leq b_i \quad , \quad i=1,2\ldots,m \quad . \tag{1.1}$$

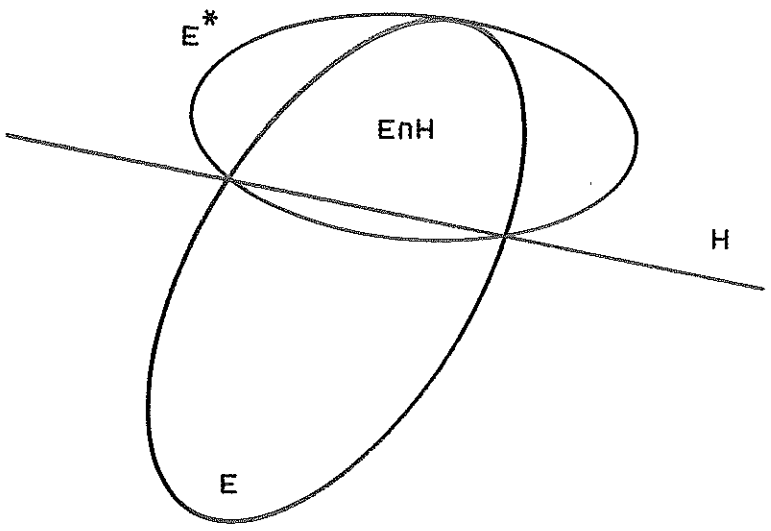


fig (1.1) Konstruktionen av ett nytt elliptiskt klot.

Man börjar då med att innesluta Q, antagen begränsad och med positiv volym, i ett elliptiskt klot E_0 med medelpunkt x_0.

Om x_0 liger utanför Q så att $a^t x_0 > b$ för någon av olikheterna i (1.1) så låter man H vara halvrummet $a^t x \leq b$ och konstruerar $E_1 = E^*$. Om istället $x_0 \in Q$ så låter man H vara halvrummet $c^t x > c^t x_0$ och konstruerar $E_1 = E^*$.

Processen fortsättes sedan på samma sätt från E_1 och man får en svit elliptiska klot $E_0, E_1, E_2 \ldots$. Man kan visa att volymen $\mathrm{vol}(E_k)$ avtar exponentiellt och att talen $c^t x_k$, där x_k är de medelpunkter i E_k som ligger i Q, konvergerar mot $\max(c^t x)$.

Khachijan använde denna metod för att avgöra om Q är tom eller inte. Under antagandet att elementen i a_i och b_i är heltal med en fast begränsning visade han att om Q inte är tom så finns ett E_0 som innehåller en icke tom del Q_0 av Q och ett tal N som beror polynomiellt på data för Q. Talet N har egenskapen att Q är tom precis då algoritmen kan fortsätta utöver N iterationer utan att medelpunkten x_k kommit in i Q.

Vi övergår nu till att beskriva algoritmen algebraiskt. Alla vektorer och matriser hänför sig till ett ortogonalt koordinatsystem i R^n. Olikheten

$$(x-x_0)^t A^{-1} (x-x_0) \leq 1 \quad , \tag{1.2}$$

där A är en symmetrisk positivt definit matris av typ n*n, framställer ett elliptiskt klot i R^n. Låt det elliptiska klotet E_k, bestämt av punkten x_k och matrisen A_k, och halvrummet H enligt ovan, sådant att x_k inte ligger i det inre av H, vara givna. Problemet är nu att uttrycka x_{k+1} och

A_{k+1} i A_k, x_k, a och b på ett sådant sätt att E_{k+1} får ovan nämnda egenskaper. För att få enklare beräkningar börjar vi med att överföra E_k resp. H i x-rummet till enhetssfären resp. halvrummet

$$y_1 \leq -\alpha \ , \ 0 \leq \alpha \leq 1$$

i y-rummet. Detta kan göras, ty det finns en reell kvadratisk matris B sådan att

$$A_k = BB^t$$

eftersom A_k är symmetrisk och positivt definit.

Variabelbytet $x - x_k = Bz$, $z \in R^n$ ger insatt i (1.2) att $z^t z = |z|^2 \leq 1$ och vridningen så att H övergår i halvrummet $y_1 \leq -\alpha$ kan göras genom variabelbytet $z = Qy$ där Q är en lämplig ortogonal och normerad matris. Matrisen B är således inte unik trots att A_k är det. Kalla matrisen för variabelbytet från x till y för B_k, dvs vi gör variabelbytet

$$x - x_k = B_k y \ . \qquad (1.3)$$

Med ledning av fig (1.2) ser vi att E_{k+1} kan skrivas

$$((y_1+c)/a)^2+(y_2/b)^2+\ldots+.(y_n/b)^2 \leq 1 \ , \ 0 \leq c \leq 1 \ . \ (1.4)$$

Kraven på E_{k+1} blir att dess rand går genom punkten $(-1,0,\ldots,0)$ på enhetssfären och har samma skärning med hyperplanet $y_1 = -\alpha$ som enhetssfären. Dessa två villkor ger ekvationerna

$$(-1+c)^2/a^2 = 1 \text{ och } (-\alpha+c)^2/a^2+(1-\alpha^2)/b^2 = 1 \ ,$$

vilka i sin tur ger

$$c = 1-a \quad , \quad b^2 = a^2(1+\alpha)/(2a-(1-\alpha)) \ .$$

Förhållandet mellan volymerna av det elliptiska klotet (1.4) och enhetssfären, som är det samma som förhållandet mellan

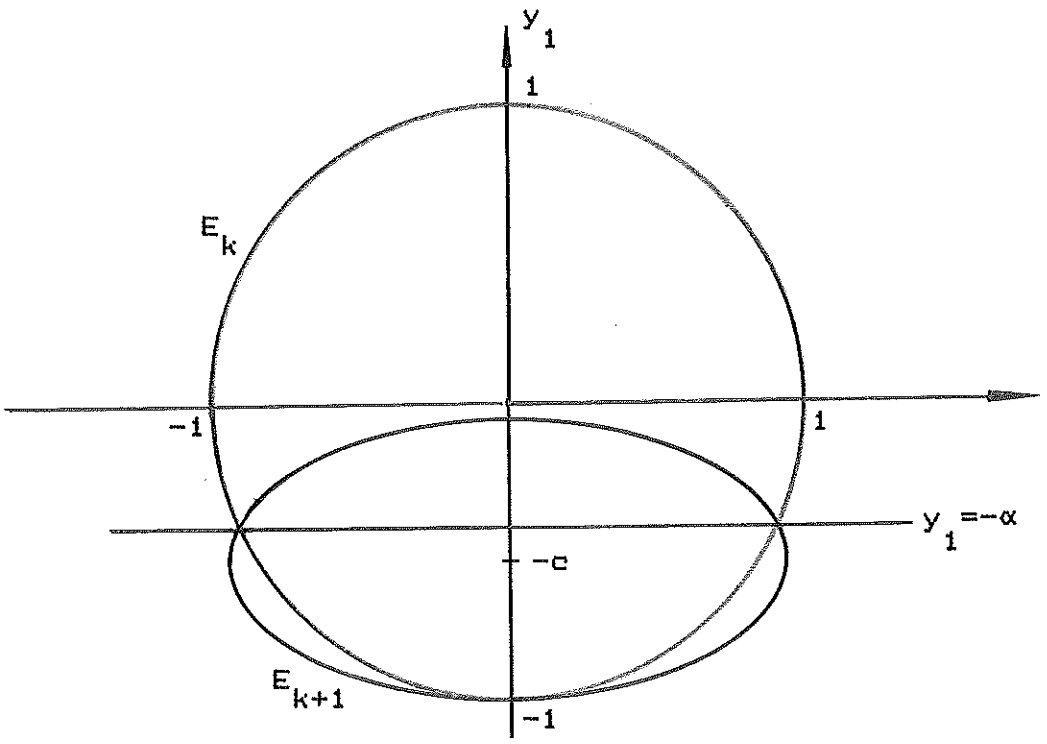


fig (1.2) Bilderna av E_k och E_{k+1} i y-rummet.

volymerna av E_{k+1} och E_k, blir

$$\mathrm{vol}(E_{k+1})/\mathrm{vol}(E_k) = ab^{n-1} = a^n[(1+\alpha)/(2a-(1-\alpha))]^{(n-1)/2}. \quad (1.5)$$

Minimering av förhållandet (1.5) med avseende på a ger

$$a = n(1-\alpha)/(n+1),\ b^2 = n^2(1-\alpha^2)/(n^2-1),\ c = (1+n\alpha)/(1+n) \quad (1.6)$$

och volymförhållandet

$$\mathrm{vol}(E_{k+1})/\mathrm{vol}(E_k) = (1-\alpha)(1-\alpha^2)^{(n-1)/2}(1+n^{-1})^{-1}(1-n^{-2})^{(1-n)/2} \leq$$

$$\leq (1-\alpha)(1-\alpha^2)^{(n-1)/2}e^{-1/2(n+1)}. \quad (1.7)$$

Skrivet i formen (1.2) får det elliptiska klotet (1.4) utseendet

$$(y-y_{k+1})\, C_{K+1}^{-1}\, (y-y_{k+1}) \leq 1 \quad (1.8)$$

där $y_{k+1} = -ce_1$, $e_1 = (1,0,\ldots,0)^t$ och

$$C_{k+1} = (n^2(1-\alpha)^2/(n^2-1))(I-2c/(1+\alpha)e_1e_1^t$$

med I som enhetsmatrisen av typ n*n och

$$c = (1+n\alpha)/(1+n) \ .$$

Det återstår nu bara att gå tillbaka till de ursprungliga koordinaterna. Uttrycket $a^t x-b$ ger

$$a^t B_k y+a^t x_k -b = d^{-1}(y_1+\alpha) = d^{-1}(e_1^t y+\alpha)$$

för något $d>0$ och alla y. Alltså är $(a^t B_k)^t=d^{-1}e_1$ varför

$$d^{-2}=(d^{-1}e_1^t)(d^{-1}e_1^t)^t=a^t B_k B_k^t a=a^t A_k a$$

och $a^t x_k -b=d^{-1}\alpha$ dvs

$$\alpha = \frac{a^t x_k -b}{(a^t A_k a)^{1/2}} \ .$$

Vidare får vi

$$x_{k+1} -x_k =B_k(y_{k+1} -y_k)=-B_k ce_1 =-cB_k(da^t B_k)^t=$$

$$=-cdA_k a=-c\frac{A_k a}{(a^t A_k a)^{1/2}} \ .$$

Likheten $y-y_{k+1} =B_k^{-1}(x-x_k)+ce_1 =B_k^{-1}(x-x_{k+1})$

ger slutligen tillsammans med uttrycket för C_{k+1}

det nya elliptiska klotet

$$E_{k+1}: \quad (x-x_{k+1})^t A_{k+1}^{-1} (x-x_{k+1}) \leq 1$$

med x_{k+1} enligt ovan och

$$A_{k+1} = B_k C_{k+1} B_k^t = -\frac{n^2}{n^2-1}(1-\alpha^2)(B_k B_k^t - B_k e_1 (B_k e_1)^t) =$$

$$= -\frac{n^2}{n^2-1}(1-\alpha^2)(A_k - \frac{2c}{1+\alpha} \cdot \frac{A_k a (A_k a)^t}{a^t A_k a}) \quad .$$

Sammanfattningsvis blir iterationsformlerna (en del parametrar är omdöpta):

$$x_{k+1} = x_k - \tau \cdot \frac{A_k a}{(a^t A_k a)^{1/2}} \quad , \qquad (1.9)$$

$$A_{k+1} = \delta (A_k - \sigma \cdot \frac{A_k a (A_k a)^t}{a^t A_k a}) \quad \text{där} \qquad (1.10)$$

$$\tau = \frac{1+n\alpha}{1+\alpha} \quad , \quad \delta = \frac{n^2}{n^2-1}(1-\alpha^2) \quad , \quad \sigma = \frac{2}{1+n} \cdot \frac{1+n\alpha}{1+\alpha}$$

(1.11)

och $$\alpha = \frac{a^t x_k - b}{(a^t A_k a)^{1/2}} \quad .$$

Slutligen är $a^t x - b$ ett av villkoren i (1.1) som inte är uppfyllt dvs

$$a^t x_k - b > 0.$$

Formlerna är som synes helt explicit givna av det elliptiska klotet E_k och ett villkor ur (1.1) som inte är uppfyllt.

Det kan i iteration k+1 hända att $\alpha > 1$. Vi ser i fig (1.2) att detta motsvarar precis det fall med tom snittmängd av E_k och halvrummet H, eller med andra ord att (1.1) saknar lösning.

1.2 KONVERGENSHASTIGHET.

I föregående avsnitt härledde vi iterationsformlerna för EM (ellopsoidmetoden). Vår härledning gav emellertid ingen som helst upplysning om hur hur många iterationer som krävs för att lösa (1.1) (eller för att avgöra om (1.1) saknar lösning). Det är klart att för att kunna bedöma metodens praktiska användbarhet så är det önskvärt att veta hur många steg som (maximalt) krävs. Dessutom bör denna gräns i möjligaste mån vara oberoende av det aktuella problemet.

Antag att samtliga koefficienter i (1.1) är heltal och definiera längden av problemet (1.1) genom

$$L = \sum_{\substack{1\le i\le m,\\ 1\le j\le n,\\ a_{ij}\neq 0}} \left[{}^{2}\log|a_{ij}| \right] + \sum_{\substack{1\le i\le m,\\ b_i\neq 0}} \left[{}^{2}\log|b_i| \right] + \tag{1.12}$$

$$+ \left[{}^{2}\log n \right] + \left[{}^{2}\log m \right] + 2mn + 2m + 4 \quad ,$$

där [x] betyder heltalsdelen av x. L är det antal bitar som krävs för att koda (1.1) i binär form.

Khachijan visade att om (1.1) har en lösning så är den innehållen i ett klot med centrum i origo och radie 2^{L}, betecknat $S(0,2^{L})$. Han visade också att (1.1) har en lösning om och endast om systemet

$$a_i^t x \le b_i + 2^{-L} \quad , \quad i=1,\ldots,m \tag{1.13}$$

har en lösning samt att om (1.1) har en lösning y så är alla punkter i klotet $S(y,2^{-2L})$ lösningar till (1.13), se ref [7].

Låt oss nu applicera EM på problemet (1.13) med $E_0=S(0,2^{L})$. Antag att vi räknar med exakt aritmetik. Hur många iterationer krävs nu (maximalt) för att avgöra om (1.1) har en lösning eller ej? Enligt ovan behöver vi inte fortsätta längre än tills $\mathrm{vol}(E_k) \le \mathrm{vol}(S(x_k,2^{-2L}))$, ty om (1.1) är lösbar så är x_k en lösning till (1.13) och om x_k inte är en

lösning till (1.13) så saknar (1.1) lösning. Med hjälp av (1.7) får vi villkoret

$$\frac{vol(E_k)}{vol(E_{k+1})} \leq e^{-k/2(n+1)} < 2^{-k/2(n+1)} < \frac{vol(S(x_k, 2^{-2L}))}{vol(E_0)} = 2^{-3L}$$

vilket ger att $k > 6n(n+1)L$. Har inte EM producerat en lösning till (1.13) efter högst

$$k = 6n(n+1)L$$

iterationer så saknar alltså (1.1) lösning.

Khachijan förfinade detta resonemang och visade att det behövs högst

$$k = 16n^2L \qquad (1.14)$$

iterationer då alla beräkningar utförs med en noggrannhet av 23L bitar före och 38nL bitar efter binärpunkten.

Detta resultat, att antalet iterationer är begränsat av ett polynom i n och L, är ett viktigt teoretiskt framsteg som när det kom väckte ett stort uppseende världen över i datalogikretsar. Man har nämligen visat att simplexmetoden, den förhärskande metoden för att lösa linjärprogrammerings-problem, för vissa problem kräver 2^n stycken iterationer för att hitta lösningen. Vidare går det utan större svårigheter att modifiera EM så att den löser fullständiga linjärprogrammeringsproblem så att antalet steg är högst polynomiellt i n och L.

Hur mycket är nu detta resultat värt i praktiken? En indikation på värdet kan vi få genom följande exempel.

Exempel: Betrakta följande problem i R^2.

$$\begin{cases} x_1 \leq 2 \\ -x_1 \leq -1 \\ x_2 \leq 2 \\ -x_2 \leq -1 \end{cases} \qquad (1.15)$$

Längden av (1.15) är L=33, varför det enligt (1.14) behövs högst 2112 iterationer för att avgöra

lösbarheten av (1.15) under förutsättning att alla beräkningar utförs med en noggrannhet av 759 bitar före och 2508 efter binärpunkten. Om vi kör problemet på en konventionell dator kommer vi naturligtvis inte i närheten av den precision som krävs.

Slutsatsen blir att för att avgöra hur effektiv EM är i praktiken är vi hänvisade till numerisk försöksverksamhet. Det kan mycket väl hända att metoden är numeriskt instabil och inte konvergerar över huvud taget.

1.3 MODIFIERINGAR AV ALGORITMEN.

I det här avsnittet kommer vi att beskriva några enkla modifieringar av EM för att öka konvergenshastigheten.

1.3.1 DJUPA SNITT.

Vi har faktiskt redan i härledningen av iterationsformlerna i avsnitt 1.1 använt oss av sk djupa snitt. Khachijan har i sin version av algoritmen translaterat det utvalda hyperplanet $a^t x=b$ till $a^t x=a^t x_k$. Nästa elliptiska klot E_{k+1} omsluter således alltid mer än hälften av E_k. Khachijans version fås ur härledningen i avsnitt 1.1 genom att sätta $\alpha=0$ överallt, se figur (1.3). I föregående avsnitt, där vi tog fram en formel för konvergenshastigheten , utnyttjade vi inte det faktum att $\mathrm{vol}(E_{k+1})/\mathrm{vol}(E_k)$ avtar monotont då $0\leq\alpha\leq 1$. Därför bör konvergenshastigheten öka om djupa snitt

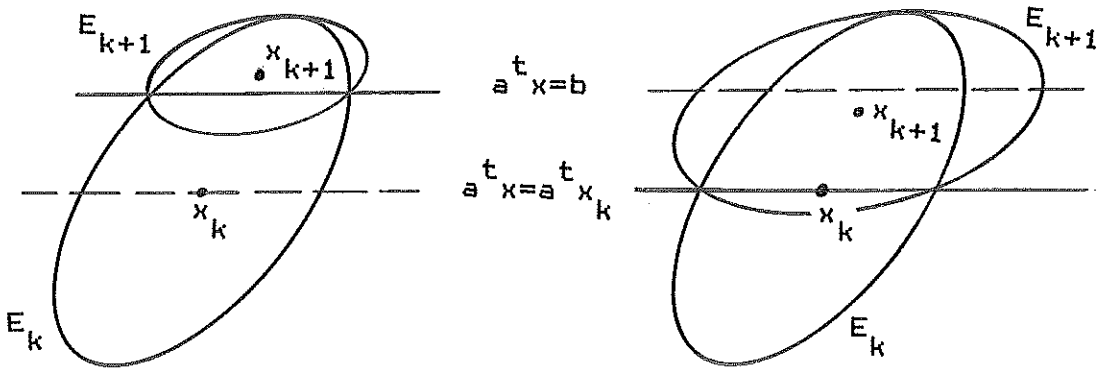


fig (1.3) Ellipsoidmetoden : a) med djupa snitt
b) Khachijans version.

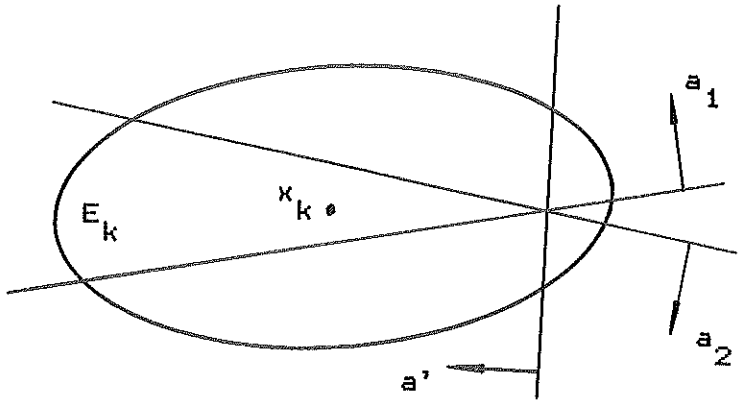
användes. Det är emellertid svårt att säga mer exakt hur mycket konvergensen förbättras. Medelvärdet av α kan ju vara väldigt nära noll trots att det djupaste snittet väljes ut i varje iteration. En tänkbar metod är att i varje iteration beräkna α för varje icke uppfylld olikhet i (1.1) och välja det djupaste snittet, dvs den olikhet som ger störst α. Det bör dock noteras att detta förfarande inte nödvändigtvis är en optimal strategi.

1.3.2 KOMBINATIONSSNITT.

Man kan ibland, genom att kombinera olikheter i (1.1), åstadkomma ett snitt som är djupare alla snitt baserade på en enda olikhet. Principen illustreras av fig(1.4). Ur beräkningssynpunkt är det emellertid tveksamt om kombinationssnitten är fördelaktiga; det kan nämligen behövas långa och tidskrävande beräkningar för att hitta ett kombinationssnitt som är djupare än det djupaste enkla snittet. Dessutom är det inte säkert att det existerar något djupare kombinationssnitt än det djupaste enkla snittet.

1.3.3 PARALLELLA SNITT.

Om (1.1) innehåller parallella par av olikheter så kan bägge olikheterna användas samtidigt för att generera ett nytt elliptiskt klot,se fig(1.5). Beteckna olikheterna med



$$a' = t_1 a_1 + t_2 a_2$$

fig (1.4) Kombination av två olikheter till ett djupare snitt.

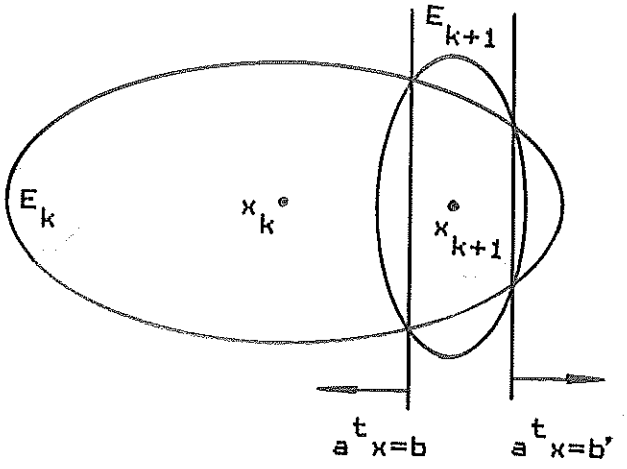


fig (1.5) Parallellt snitt.

$$a^t x \leq b \text{ och } -a^t x \leq -b' \text{ där } a^t x_k > b$$

och definiera α och α' genom

$$\alpha = \frac{a^t x_k - b}{(a^t A_k a)^{1/2}} \quad \text{och} \quad \alpha' = \frac{b' - a^t x_k}{(a^t A_k a)^{1/2}} \quad .$$

Då ger formlerna (1.9)-(1.10) tillsammans med

$$\sigma = \frac{1}{n+1}\left(n + \frac{2}{(\alpha-\alpha')^2}(1-\alpha\alpha'-\varrho/2)\right) ,$$

$$\tau = \frac{(\alpha-\alpha')\sigma}{2} \quad , \qquad (1.16)$$

$$\delta = \frac{n^2}{n^2-1}(1-(\alpha^2+\alpha'^2-\varrho/n)/2) \quad ,$$

$$\text{där } \varrho = (4(1-\alpha^2)(1-\alpha'^2) + n^2(\alpha^2-\alpha'^2)^2)^{1/2} \quad ,$$

det elliptiska klotet med minsta volym som innehåller snittmängden av E_k och $\{x \in R^n \mid b \leq a^t x \leq b'\}$. Formlerna gäller

under förutsättning att $0 \leq \alpha \leq -\alpha' \leq 1$.

Har man i sitt problem många parvis parallella olikheter bör algoritmen snabbas upp avsevärt. Om b=b', dvs om vi har ett likhetsvillkor så minskar rangen av A_k med ett och E_{k+1} blir "platt" i riktning a.

1.4 LINJÄR PROGRAMMERING MED ELLIPSOIDMETODEN.

I detta avsnitt beskriver vi några olika metoder att lösa linjärprogrammeringsproblem (LP-problem) med ellipsoid-metoden. LP-problem kan i allmänhet formuleras så här:

Maximera $c^t x$ under bivillkoren

$$A^t x \leq b \quad , \qquad (1.17)$$

$$x_i \geq 0 \quad , \quad i = 1,\ldots,n$$

där c är en n-dimensionell kolonnvektor, A är en n*m-matris och b är en m-dimensionell kolonnvektor.

Utrycket $A^t x \leq b$ är bara ett kompakt sätt att skriva (1.1). Den linjära funktion, $c^t x$, som ska optimeras kallas för objektfunktionen eller bara objektet.

1.4.1 ANVÄNDANDE AV DUALEN.

Det duala problemet till (1.17) lyder:

Minimera $b^t y$ under bivillkoren

$$Ay \leq c \ , \qquad (1.18)$$

$$y_i \geq 0 \ , \ i=1,\ldots,m$$

Dualitetssatsen säger att (1.17) har en begränsad lösning om och endast om detsamma gäller för (1.18) och i så fall är $\max(c^t x) = \min(b^t y)$. Dessutom gäller för varje x och y som uppfyller sina respektive bivillkor att $c^t x \leq b^t y$.

Problemen (1.17) och (1.18) kan alltså lösas samtidigt genom att lösa problemet

$$\begin{cases} A^t x \le b \\ -x \le 0 \\ -Ay \le -c \\ -y \le 0 \\ -c^t x + b^t y \le 0 \end{cases} \qquad (1.19)$$

En nackdel med denna metod är att (1.19) har dimensionen n+m. Dimensionen har alltså ökat med m i förhållande till det ursprungliga problemet (1.17). Detta minskar naturligtvis konvergenshastigheten. Vidare har man i praktiska LP-problem ofta explicit givna begränsningar på variablerna. Dessa gränser bör då utnyttjas för att bestämma E_0, om volymen då blir mindre än volymen av $S(0,2^L)$. Men det kan vara mycket svårt att bestämma begränsningar på de motsvarande duala variablerna, varför volymen av E_0 i R^{n+m} kan bli mycket större än nödvändigt. Observera att lösningen till (1.19) ligger i hyperplanet $c^t x = b^t y$, vilket innebär att lösningsmängden har volymen 0. Det är därför nödvändigt att lägga på en störning på högerledet i (1.19), t.ex. $-c^t x + b^t y \le \varepsilon$ där $\varepsilon > 0$.

1.4.2 INTERVALLHALVERING.

I denna metod förutsätter vi att polyedern Q, definierad av bivillkoren i (1.17), är begränsad med ett icke-tomt inre, dvs att $vol(Q) > 0$. Om det på förhand är känt att $vol(Q) = 0$ kan metoden användas efter en lämplig störning av bivillkorens högerled.

Vi börjar med att finna en punkt, x', i det inre av Q med EM. Nu är $\xi_u = c^t x'$ en nedre och $\xi_ö = c^t x' + (c^t A' c)^{1/2}$ en övre begränsning till $\max(c^t x)$ (x' och A' definierar det sist bestämda elliptiska klotet. I fortsättningen förfar vi enligt följande iterativa procedur:

Steg 1) Om $\xi_{ö}-\xi_{u}<\varepsilon$, där ε är en förutbestämd noggrannhet, så stoppa, annars gå till steg 2).

Steg 2) Sätt $\xi=(\xi_{ö}+\xi_{u})/2$ och lös med EM problemet (1.17) tillsammans med villkoret $-c^{t}x\leq-\xi$.

Steg 3) fall a) Problemet i steg 2) gav lösningen x'. Spara x' och $c^{t}x'$. Sätt $\xi_{u}=c^{t}x'$ och gå till steg 1).
fall b) Problemet i steg 2) saknar lösning. Sätt $\xi_{ö}=\xi$ och fortsätt till steg 1).

När en huvuditeration startar med ett ξ större än det gamla kan steg 2) startas upp med E_{0} lika med det sista elliptiska klotet från föregående huvuditeration. Om istället det nya ξ är mindre än det gamla så kan EM startas upp med det senast beräknade elliptiska klotet vars centrum löser problemet i steg 2).

Fördelen med intervallhalveringsmetoden är att den hela tiden arbetar i R^{n}. Ur praktisk synpunkt är det emellertid ofördelaktigt att behöva applicera EM på problem som saknar lösning, dels eftersom mittpunkt och matris för ett extra elliptiskt klot måste lagras undan, och dels eftersom det kan ta många iterationer innan det kan avgöras att ett problem saknar lösning.

1.4.3 METODEN MED OBJEKTSNITT.

Förutsättningarna är desamma som i intervallhalverings-metoden.

Vi börjar även här med att finna en punkt x_{k} i Q. Varje gång som EM ger en punkt i Q använder vi oss av objektsnittet $-c^{t}x\leq-c^{t}x_{k}$, se fig (1.6). Denna metod är troligen den mest effektiva för praktiskt bruk. den arbetar hela tiden med lösbara system i R^{n}, åtminstone om problemet från början har en lösning. Övre begränsning till $\max(c^{t}x)$ blir

$$\xi_{k,ö}=\min[\xi_{k-1,ö},c^{t}x_{k}+(c^{t}A_{k}c)^{1/2}]$$

och den nedre begränsningen är

$$\xi_{k,u}=\max[\xi_{k-1,u},c^{t}x_{k}]$$

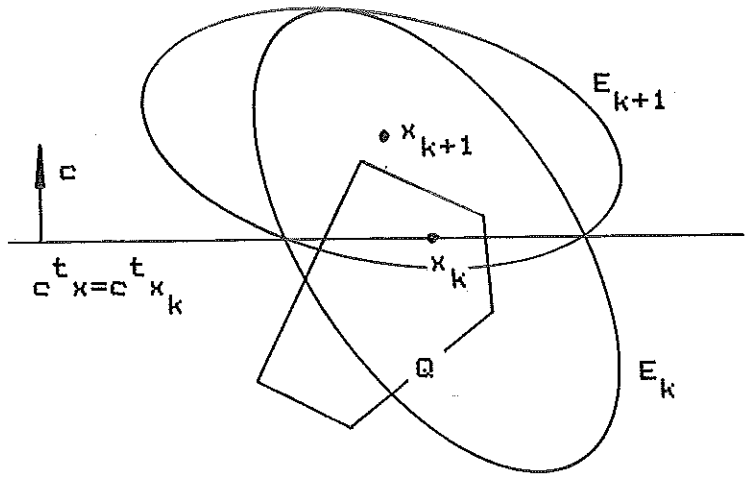


fig (1.6) Objektsnitt.

Begränsningarna uppdateras endast då $x_k \in Q$. Vi kan således stoppa metoden då $\xi_{ö} - \xi_{u} < \varepsilon$ där ε är den önskade noggrannheten.

1.5 IMPLEMENTERING.

När vi försöker implementera EM på någon konventionell dator dyker ett nytt problem upp. Formlerna (1.9)-(1.11) kräver i princip exakta beräkningar medan alla konventionella datorer har en ändlig ordlängd och således endast tillåter approximativa beräkningar. Metoden måste alltså ges en sådan formulering att avrundningsfelen inte fortplantas alltför våldsamt. Kort sagt: iterationsformlerna för EM måste implementeras på ett numeriskt stabilt sätt.

Om formlerna (1.9)-(1.11) användes direkt för att implementera EM så kommer matriserna A_k (nästan) alltid att bli indefinita, pga avrundningsfelen. Detta leder i sin tur till att talet $a^t A_k a$, vars kvadratrot behövs, kan bli noll eller negativt.

Ett sätt att undvika sådana numeriska problem är att representera det elliptiska klotet E_k med dess centrum x_k och en icke-singulär matris Q_k som transformerar E_k till enhetsklotet med centrum i origo. En sådan matris finns alltid, jmf avsnitt 1.1, och

$$A_k = Q_k Q_k^t \quad . \tag{1.20}$$

Substituerar vi (1.20) i formlerna (1.9)-(1.10) så får vi

$$x_{k+1} = x_k - \tau Q_k \omega_k \quad \text{där} \quad \omega_k = \frac{Q_k^t a}{||Q_k^t a||} \tag{1.21}$$

och || || är den euklidiska normen:

$$||Q|| = (\Sigma q_{ij}^2)^{1/2} \ ,$$

samt $A_{k+1} = \delta(Q_k Q_k^t - \sigma(Q_k \omega_k^t)(Q_k \omega_k^t)^t) =$

$$= \delta Q_k (I - \sigma \omega_k \omega_k^t) Q_k^t =$$

$$= \delta Q_k (I - \pi \omega_k \omega_k^t)(I - \pi \omega_k \omega_k^t) Q_k^t$$

där $\quad (1-\pi) = \pm(1-\sigma)^{1/2}$.

Detta ger att

$$Q_{k+1} = \delta^{1/2} Q_k (I - \pi \omega_k \omega_k^t).$$

Från avsnitt (1.1) vet vi att Q_k kan ersättas med $Q_k J_k$ där J_k är ortogonal, utan att det påverkar A_k. Multiplikation med J_k svarar mot en vridning och om vi låter ω_k vara den första kolumnen i J_k så får vi

$$Q'_{k+1} = \delta^{1/2} Q_k (I - \pi\omega_k \omega_k^t) J_{k+1} =$$

$$= \delta^{1/2} Q_k (I - \pi\omega_k e_1^t) = Q_k J_{k+1} D_k \qquad (1.22)$$

där $e_1 = (1\ 0 \ldots\ldots 0)^t$ och

$$D_k = \mathrm{diag}(\delta^{1/2}(1-\pi), \delta^{1/2}, \ldots\ldots, \delta^{1/2}) \ .$$

Formel (1.21) tillsammans (1.22) är de uppdateringsformler som Khachijan gav i sin version av EM, bortsett från en faktor 2^{Δ}, $\Delta = 1/8n^2$, i (1.22). Faktorn 2^{Δ} introducerade Khachijan för att kompensera för avrundningsfel. Anledningen till att (1.21) och (1.22) lämpar sig bättre för implementering än (1.9)-(1.10) är att faktorn $(a^t A_k a)^{1/2}$ övergår i normen i (1.21) som alltid är icke-negativ. På grund av avrundningsfel kan emellertid Q_k tappa rang, vilket kan få till följd att normen blir noll. Detta är ju inte heller så lyckat eftersom normen ingår i nämnaren i (1.21). Om en sådan situation uppstår kan algoritmen fortsätta, under förutsättning att $Q_0 = \eta I$, om den senast beräknade matrisen Q_k ersätts med $Q_k = \kappa Q_0$, där κ är produkten av alla succesivt beräknade δ^2, se ref [4].

Det är också möjligt att använda faktoriseringen

$$A_k = L_k D_k L_k^t$$

där L_k är en vänster triangulär matris med ettor i diagonalen och D_k är en positivt definit diagonalmatris. Detta betyder att det krävda minnesutrymmet minskar avsevärt. Dessutom behövs det färre räkneoperationer för att uppdatera L_k och D_k. LDL-faktoriseringen finns beskriven i ref. [4] där också numeriskt stabila uppdateringsalgoritmer för L och D finns.

1.6 ICKE-LINJÄRA OBJEKTFUNKTIONER.

I beskrivningen av EM har vi sällan utnyttjat att objektfunktionen är linjär. Faktum är att EM kan användas för optimera även icke-linjära funktioner, åtminstone om metoden med objektsnitt används, se 1.4.3 . Ett tillräckligt villkor är att objektfuntionen är konvex. Detta innebär närmare bestämt att en funktion f(x) kan minimeras (maximeras) om alla områden av typen $\{x\in R^n \mid f(x)\leq(\geq)f(x_k)\}$ är konvexa. För att kunna använda EM på en sådan funktion måste en subgradient g till f beräknas varje gång man hamnat i en punkt x_k som uppfyller bivillkoren. Därefter kan metoden med objektsnitt användas med objektsnittet $g^t x\leq(\geq)g^t x_k$. EM utvecklades ursprungligen för att användas inom området konvex programmering.

1.7 ELIMINERING AV LIKHETSVILLKOR.

LP-problem innehåller ofta likhetsvillkor. Genom att lägga på en störning kan EM hantera problemet direkt. Tråkigt nog är detta förfarande inte så lyckat. Lösningsmängden blir nämligen mycket "smal" i alla normalriktningar till hyperplanen, definierade av likhetsvillkoren, och mycket utsträckt i de riktningar som är parallella med samtliga hyperplan. Och detta faktum kan leda till att de elliptiska kloten genererade av EM blir mycket "ovala". Matriserna A_k innehåller då både mycket stora och mycket små tal, vilket lätt kan orsaka numeriska problem. Det finns dock en utväg ut ur problemet.

Betrakta problemet

$$A^t x = b \quad , \; 0\leq x\leq v \tag{1.23}$$

där A är av typ n*m och har rang m. (1.23) kan reduceras till ett problem i n-m variabler. I det följande beskriver vi hur reduktionen kan göras på ett sätt som är numeriskt stabilt. Beskrivningen finns även i ref. [4] sid 28-29. Matrisen A kan skrivas

$$A = [Q_1|Q_2]\begin{bmatrix} R \\ 0 \end{bmatrix} = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$$

där Q är en ortogonal n*n matris, Q_1 har m och Q_2 har n-m kolonner, och R är av typ m*m. Kolonnerna i Q_2 är en ortogonal och normerad bas för rummet $L=\{x\in R^n \mid A^t x=0\}$ och Q_1:s kolonner är det ortogonala komplementet till L. Q_2 ska mao

uppfylla $A^t Q_2=0$ och $Q_1^t Q_2=0$.

Definiera nu $y_1\in R^m$ och $y_2\in R^{n-m}$ genom

$$x = [Q_1|Q_2]\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = Q_1 y_1 + Q_2 y_2 \qquad (1.24)$$

Instoppning av (1.24) i (1.23) ger

$$b = A^t x = A^t Q y = R^t y_1 .$$

Variblerna y_1 kan således beräknas en gång för alla.

Linjära former $a^t x$ övergår i

$$a^t x = a^t(Q_1 y_1 + Q_2 y_2) = (Q_2^t a)^t y_2 + (Q_1^t a)^t y_1 .$$

Vidare övergår villkor av typen $0\leq x\leq v$ i

$$-Q_1 y_1 \leq Q_2 y_2 \leq v - Q_1 y_1$$

Därmed är reduktionen klar.

2 PROGRAMMERING AV ELLIPSOIDMETODEN.

Efter föregående kapitels teoretiska beskrivning av EM är vi preparerade för att konstruera ett datorprogram för algoritmen. I detta kapitel ges ett exempel på nedbrytning av problemet i delproblem. Naturligtvis kan detta göras på många olika sätt; iterationsformlerna kan implementeras olika och ingredienserna i kap 1 kan kombineras på olika sätt.

2.1 FORUTSÄTTNINGAR.

Programmet ska med EM lösa följande problem:

Minimera $f(x)=a_0^t x$, $x\in R^n$, under bivillkoren

$$b_i - c_i \le a_i^t x \le b_i + c_i \quad , \; c_i \rangle 0 \; , \; i=1,\ldots,m_1 \; , \tag{2.1}$$

$$a_i^t x \le b_i \quad , \; i=m_1+1, m_1+2, \ldots, m_2 \; .$$

Min(f(x)) ska beräknas med önskad noggrannhet. Metoden med objektsnitt ska användas, se 1.4.3 . När uppdateringen av (x_k, Q_k) baseras på ett parallellt par av olikheter ska tekniken med parallella snitt användas, se 1.3.3 , om så är möjligt.

Uppräkningsformlerna (1.9)-(1.10) implementeras med faktoriseringen $A_k = Q_k Q_k^t$, se avsnitt 1.5.

De formler som ska programmeras blir, med en del parametrar omdöpta (jmf (1.9)-(1.11) och (1.22)):

$$x_{k+1} = x_k - \tau Q_k \omega_k \quad , \tag{2.2}$$

$$Q_{k+1} = Q_k J_{k+1} D_k \quad , \tag{2.3}$$

$$\omega_k = Q_k^t a/\gamma \quad , \quad \gamma = ||Q_k^t a|| \quad , \tag{2.4}$$

$$D_k = \mathrm{diag}(\pi,\delta,\delta,\ldots,\delta) \quad ,$$

J_{k+1} är en ortogonal matris vars första kolonn är ω_k,

$$\tau = \frac{1+n\alpha}{1+n}k \quad , \quad \pi = \frac{n}{n+1}(1-\alpha) \quad , \quad \delta = \frac{n}{(n^2-1)^{1/2}}(1-\alpha^2)^{1/2}$$

$$\text{där} \quad \alpha = (a^t x_k - b)/\gamma \quad , \quad 0 \le \alpha \le 1 \quad . \tag{2.5}$$

Om uppdateringen av x_k och Q_k baseras på ett parallellt par av olikheter blir formlerna för τ,π,δ istället (jmf avsnitt 1.3.3)

$$\tau = \frac{(\alpha-\alpha')\sigma}{2} \quad , \quad \delta = \frac{n}{(n^2-1)^{1/2}}(1-(\alpha^2+\alpha'^2-\varrho/n)/2)^{1/2} \quad ,$$

$$\pi = \delta(1-\sigma)^{1/2} \quad \text{där} \tag{2.6}$$

$$\sigma = \frac{1}{n+1}(n + \frac{2}{(\alpha-\alpha')^2}(1-\alpha\alpha'-\varrho/2)) \quad ,$$

$$\varrho = [4(1-\alpha^2)(1-\alpha'^2) + n^2(\alpha^2-\alpha'^2)^2]^{1/2} ,$$

$$\alpha = (a^t x_k - b - c)/\gamma \quad \text{och} \quad \alpha' = (b - c - a^t x_k)/\gamma$$

där $\alpha \le -\alpha' \le 1$.

Formlerna är helt explicita förutom att matrisen J_{k+1} inte har specificerats. J_{k+1} väljes som (se ref. [3])

$$J_{k+1} = \begin{bmatrix} \omega_1 & R_2 & 0 & \cdots & 0 & \cdots & 0 \\ \omega_2 & -\frac{\omega_1\omega_2}{R_2} & \frac{R_3}{R_2} & & 0 & & 0 \\ \omega_3 & -\frac{\omega_1\omega_3}{R_2} & & & 0 & \text{(i-2) 0:or} & 0 \\ \vdots & \vdots & & & \vdots & & \vdots \\ & & & & 0 & & \\ & & & & \frac{R_i}{R_{i-1}} & & \\ & & & & -\frac{\omega_{i-1}\omega_i}{R_{i-1}R_i} & & \\ & & & & \vdots & & \vdots \\ \omega_n & -\frac{\omega_1\omega_n}{R_2} & \cdots & \cdots & -\frac{\omega_{i-1}\omega_n}{R_{i-1}R_i} & \cdots & -\frac{\omega_{n-1}\omega_n}{R_{n-1}R_n} \end{bmatrix}$$

Här är $(\omega_1,\ldots,\omega_n)^t = Q_k^t a/\gamma$ och

$R_i^2 = \omega_i^2 + \ldots + \omega_n^2$ om $\omega_n \neq 0$.

Om $\omega_n = \omega_{n-1} = \ldots = \omega_{n-p+1} = 0$ och $\omega_{n-p} \neq 0$ ersätts n av $n-p$ i formeln för R_i och enhetsmatrisen av typ $p*p$ sätts in i nedre högra hörnet av J_{k+1}.

2.2 LAGRING DATA OCH MATRISER.

I allmänhet innehåller vektorerna a_i i (2.1) till största delen nollor. Detta kan utnyttjas för att spara minnesutrymme genom att endast lagra element skilda från noll. Vi utnyttjar detta på följande vis:

Alla vektorer a_i lagras i en vektor A. Endast element skilda från noll lagras. För att nå element nummer J i vektorn a_i behövs två accessvektorer, AROW och ACOL. AROW[i] anger på vilken plats i A som vektorn a_i börjar. Vektorn a_i finns alltså lagrad på platserna AROW[i] tom AROW[i+1]-1 i A. ACOL[j] innehåller ordningsnummer i vektorerna a_i för element nummer J i A. Om A[k]=a_{ij}, där a_{ij} är det j:te elementet i a_i, så är ACOL[k]=j.

I varje iteration ska den ortogonala matrisen J_{k+1} beräknas. Som synes på sid. 25 innehåller den endast nollor i övre högra hörnet. Dessa behöver inte heller lagras. Matrisen J_{k+1} D_k lagras kolumnvis i vektorn JD. En accessvektor ACJD behövs för att markera början på ny kolumn. ACJD[i] anger platsen i JD för det sista elementet i kolumn nummer i-1. Om J_{k+1} D_k är av typ n*n så får ACJD värdena

ACJD[1]=0

ACJD[i]=(i-1)n-(i-2)(i-3)/2 , i=2,3,....,n .

2.3 UPPDELNING I DELPROBLEM.

Iteration k+1 delar vi upp i två delar:

1) Välj ut ett snitt bland bivillkoren, dvs välj ut ett villkor ur (2.1) som inte är uppfyllt. Om alla bivillkoren är uppfyllda , så undersök om objektet har beräknats tillräckligt noggrannt. Har det det så ska en flagga sättas som indikerar att vi är klara.

2) Uppdatera x_k och Q_k enligt (2.2)-(2.3).

Del 1) och 2) låter vi vara underprogram benämnda CUTELLIPSOID och NEWELLIPSOID. Vidare behövs det göras en del initieringar innan itererandet kan påbörjas. Vi samlar dessa i ett underprogram INITIATE. Programstommen får nu utseendet:

```
procedure LPEM.....................;
    procedure INITIATE.............;
    procedure CUTELLIPSOID.........;
    procedure NEWELLIPSOID.........;
begin
    INITIATE;
    CUTELLIPSOID;
    while not READY do
    begin
        NEWELLIPSOID;
        CUTELLIPSOID;
    end;
end;
```

LPEM är här namnet på hela programmet (som vi har valt att lägga som en procedur). READY är en logisk variabel som sätts till true (i CUTELLIPSOID) då algoritmen termineras.

2.2.1 UNDERPROGRAMMET CUTELLIPSOID.

Här ska först ett snitt väljas ut, varefter ω, γ, τ, δ och π beräknas enligt formlerna (2.4)-(2.6). Som snitt väljes det först påträffade bivillkoret som inte är uppfyllt. Om samtliga bivillkor är uppfyllda testar vi om objektets värde ligger inom den fastställda noggrannheten. Gör det inte det så väljes objektsnittet

$$a_o^t x \leq a_o^t x_k .$$

Låt oss nu i grova drag se hur ovanstående översättes till programvara. Det är lämpligt att beräkna

$$f_i = a_i^t x_k - b_i$$

och använda f_i för att testa om villkor nummer i är uppfyllt. Vidare kan varje f_i beräknas och testas i en repetitionsslinga som vi, enligt ovan, väljer att lämna så snart ett villkor inte är uppfyllt, eller då samtliga villkor testats och befunnits vara uppfyllda. Om det sistnämnda inträffar går vi vidare och testar objektet enligt ovan. Urvalsdelen av CUTELLIPSOID får det schematiska utseendet:

```
procedure SELECTACUT;
    procedure TESTPARALLELCONSTRAINT....;
    procedure TESTCONSTRAINT............;
    procedure TESTOBJECT................;
begin
    .
    .
    while not FOUND and (villkor finns kvar att testa) do
    begin
        .
        beräkna f
                 i
        .
        if i≤m  then
              i
            TESTPARALLELCONSTRAINT
        else
            TESTCONSTRAINT;
    end;
    if not FOUND then
        TESTOBJECT;
end;
```

Vi har här valt att lägga testerna i två skilda procedurer eftersom testerna, beroende på typen av villkor, blir något olika (då i≤m testar vi två villkor på en gång).
1

I proceduren TESTOBJECT beräknas objektets värde OBJ för punkten x_k samt övre och undre begränsningar för objektet.

Övre och undre gräns beräknas enligt (se avsnitt 1.4.3)

OBJMAX:= min(OBJ,OBJMAX) och

OBJMIN:= max(OBJ-GAMMA,OBJMIN)

där GAMMA är lika med γ i (2.4) för $a=a_0$ och övriga beteckningar är självklara. Därefter testas om objektet har beräknats tillräkligt noggrannt. Om så är fallet sätts READY till true.

När ett snitt har valts ut i SELECTACUT ska vektorn ω och parametrarna γ, π och δ beräknas enligt formlerna på sid 25 . Eftersom γ ingår i beräkningen av OBJMIN i proceduren TESTOBJECT lägger vi beräkningen av γ och ω i en egen procedur som kan anropas från TESTOBJECT. Beräkningarna är rakt på sak och behöver ej förklaras i detalj. Dock bör det nämnas att om det utvalda villkoret är ett parallellt par av olikheter så bör detta noteras i programmet, lämpligen genom att sätta en logisk variabel i TESTPARALLELCONSTRAINT, samt att varje beräknat α ska testas om $\alpha>1$, ty om så är fallet så saknar problemet lösning. Dessutom är det lämpligt att kontrollera att $\gamma>0$ innan division med γ utföres (se avsnitt 1.5).

2.2.2 UNDERPROGRAMMET NEWELLIPSOID.

Detta underprogram delar vi upp i två delar, NEWX och NEWQ, där uppdateringen av x_k och Q_k göres. Stommen blir då:

```
procedure NEWELLIPSOID;
    procedure NEWX.............;
    procedure NEWQ.............;
begin
    NEWX;
    NEWQ;
end;
```

I proceduren NEWX beräknas x_{k+1} enligt formel (2.2) och i proceduren NEWQ beräknas först J_{k+1} D_k enligt sid. 24-25, varefter Q_{k+1} beräknas enligt (2.3).

Därmed är vi mogna att presentera ett färdigt program. Ett sådant, LPEM, finns i appendix B. I appendix A finns en bruksanvisning för LPEM.

3. TESTEXEMPEL.

I det här kapitlet redovisar vi resultatet av datorkörningar av LPEM på två testexempel. Datorn är en VAX 11/780. Vi börjar med

3.1 KLEE-MINTYS PROBLEM.

Klee-Mintys problem lyder (se ref. [1]):

Minimera $f(x)=c^t x$, $x \in R^n$, under bivillkoren

$$\begin{aligned} a_i^t x &\leq b_i \quad , i = 1,2,.....,n \\ x_i &\geq 0 \quad , i = 1,2,.....,n \end{aligned} \tag{3.1}$$

där $c = [10^{n-1} \;\; 10^{n-2} \;\; ..... \;\; 10^1 \;\; 0]^t$,

$a_1 = [1 \;\; 0 \;\; ..... \;\; 0]^t$,

$a_i = [2 \cdot 10^{i-1} \;\; 2 \cdot 10^{i-2} \;\; ..... \;\; 2 \cdot 10^1 \;\; 1]$, $i = 2, ...,n$

och $b_i = 10^{2(i-1)}$, $i = 1, ...,m$.

Det intressanta med detta problem är att det är ett av de få kända problem där Simplex-metoden kräver ett exponentiellt ökande (i n) antal iterationer, 2^n-1 styck, se ref [1]. Därför kan det vara av intresse att se hur ellipsoidmetoden klarar detta problem (EM ska teoretiskt kräva endast ett polynomiellt antal iterationer, se avsnitt 1.2).

Lösningen till (3.1) är $\min(f(x))=-10^{-2(n-1)}$ med lösningsvektorn $x = [0 \;\; ....0 \;\; 10^{2(n-1)}]^t$.

Som startvärden till LPEM väljer vi därför

$x_0 = [0 \;\; .....0]^t$ och

$Q_0 = \text{diag}[q_1, ...,q_n]$ där

$$q_i = 1.01 \cdot n^{1/2} \cdot 10^{2(n-1)} \quad , \; i = 1, \ldots\ldots, n \; .$$

Observera att detta ger en volym på det första elliptiska klotet som är mycket mindre än det val av startellipsoid som Khachijan anger. Detta favoriserar naturligtvis resultatet till EM:s fördel.

Vi väljer att beräkna min(f) med noggrannheten $\varepsilon = 10^{2(n-1)-6}$. Detta bör ge åtminstone 5 korrekta siffror i min(f).

Resultatet från körningarna ses i tabell 3.1. I samtliga fall hittade LPEM rätt lösning med 6 korrekta siffror.

Tabell 3.1 Resultat från körningar av Klee-Mintys problem.

Antal variabler	Exekveringstid	Antal iterationer	
	CPU-sekunder	LPEM	Simplex-metoden
2	0.18	99	3
3	0.66	226	7
4	1.73	413	15
5	3.64	616	31
6	7.20	888	63

Vi noterar att LPEM kräver mångdubbelt fler iterationer än Simplex-metoden. Mera uppmuntrande är att ökningstakten för antalet iterationer tycks vara långsammare för LPEM. I diagrammet i fig (3.1) ser vi en klar antydan om att antalet iterationer för LPEM växer endast polynomiellt.

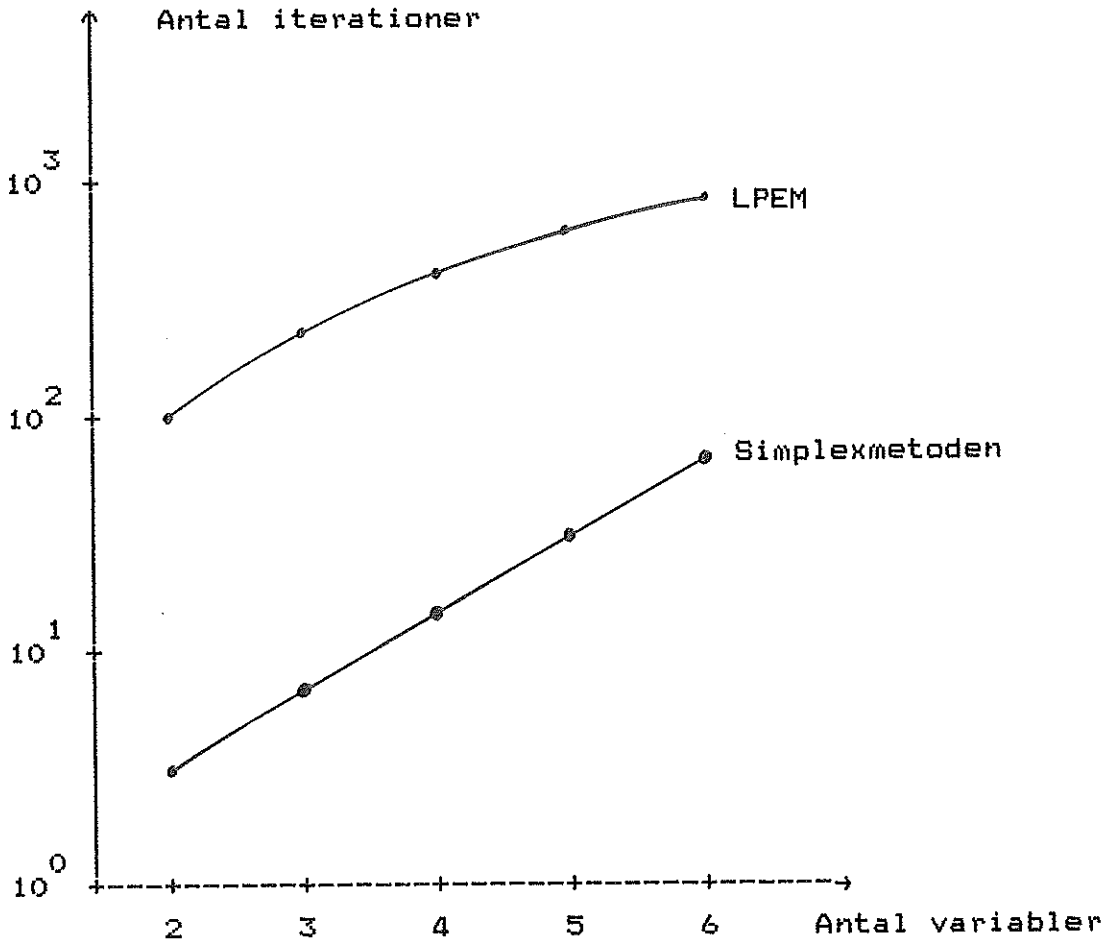


fig 3.1 Diagram över antalet iterationer för att lösa Klee-Minty's problem som funktion av antalet variabler.

3.2 LP-REGULATORN.

Vi övergår nu till att testa LPEM på ett större exempel, nämligen på LP-regulatorn. LP-regulatorn finns beskriven i Per-Olof Gutmans doktorsavhandling CONTROLLERS FOR BILINEAR CONSRTAINED LINEAR SYSTEMS, ref [5], varför vi här nöjer oss med en mycket kort beskrivning och koncentrerar oss på det optimeringsproblem som regulatorn ger.

3.2.1 KORT BESKRIVNING AV LP-REGULATORN.

LP-regulatorn förutsätter att det system som ska styras finns beskrivet som ett linjärt samplat system med begränsningar på både styr- och tillståndsvariablerna:

$$x(t+T) = \phi\, x(t) + \Gamma\, u(t) \;,\; x \in R^n,\; u \in R^m \;,$$

$$\alpha_{u(j)} \le u_j(t) \le \beta_{u(j)} \;,\; j = 1,2,\ldots,m \;,\; \forall\, t \ge t_0 \qquad 3.2)$$

$$\alpha_{x(i)} \le x_i(t) \le \beta_{x(i)} \;,\; i = 1,2,\ldots,n \;,\; \forall\, t \ge t_0 \;.$$

ϕ och Γ är matriser av typ n*n och n*m respektive.

Syftet är att driva tillståndsvektorn från dess initialvärde till en förutbestämd punkt, som definierar origo i tillståndsrummet, på så kort tid som möjligt. Detta kan göras genom att ansätta en tidshorisont τ, där τ är en gissning av hur många samplingsintervall som behövs för att driva tillståndsvektorn till det rätta värdet, och minimera en förlustfunktion

$$\hat{\ell} = |x(\tau)|_{\hat{c}} = \sum_{i=1}^{n} c_i |x_i(\tau)|$$

där $c = [\, c_1, c_2, \ldots, c_n \,]$,

samt iterera över τ tills det minsta τ för vilket $\min(\hat{\ell}) < \varepsilon$, där ε är en förutbestämd testkvantitet, har beräknats.

Det kan emellertid hända att någon tillståndsvariabel i verkligheten har hamnat utanför någon av sina begränsningar. I så fall är det möjligt att det inte finns någon tillåten styrsignal som driver tillbaka tillståndet till det tillåtna området på ett samplingsintervall och därmed kan inte LP-regulatorn producera en tillåten lösning för något τ. För att förhindra tillståndsvariablerna från att passera sina

begränsningar inför Gutman en sk "cordon sanitaire" genom att addera till ett extra straff i förlustfunktionen för de tillståndsvariabler som ligger i närheten av det förbjudna området. Förlustfunktionen, som ska minimeras, blir då istället

$$\hat{\ell} = \sum_{t=1}^{\tau-1} \sum_{i=1}^{n} \left[r_i \max[-x_i(t)+\gamma_x(i),0] + w_i \max[x_i(t)-\delta_x(i),0] \right] + \\ + \sum_{i=1}^{n} c_i |x_i(\tau)| \quad , \tag{3.3}$$

där $r = [\, r_1, r_2, \ldots, r_n \,]$ och pss för γ_x^t, w, δ_x^t och c.

Som stoppkriterium användes $|x(\tau)|_d < \varepsilon$.

Problemet som ska lösas för givet τ blir:

Minimera $\hat{\ell}$ enligt (3.3) under bivillkoren

$$Az = b \, , \; s \le z \le v \quad \text{där}$$

$$A = \begin{bmatrix} I & -\Gamma & -\phi & 0 & 0 & \cdots & 0 \\ 0 & 0 & I & -\Gamma & -\phi & 0 & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & I & -\Gamma \end{bmatrix} \quad , \tag{3.4}$$

$$z = [\, x(t)^t \; u(t-1)^t \; \ldots \; x(1)^t \; u(0)^t \,]^t \, , \; z \in R^{\tau(n+m)} \, ,$$

$$b = [\, 0 \; 0 \; \ldots \; 0 \; (\phi x(0))^t \,]^t \, ,$$

$$s = [\, \alpha_x^t \; \alpha_u^t \; \ldots \; \alpha_x^t \; \alpha_u^t \,]^t \text{ och}$$

$$v = [\, \beta_x^t \; \beta_u^t \; \ldots \; \beta_x^t \; \beta_u^t \,]^t \, .$$

I drift läses det aktuella tillståndet, x(0), vid varje samplingsintervall in till LP-rutinen, varefter problemet (3.4) löses för olika τ tills det minsta τ ($=\tau_{opt}$) för vilket stoppkriteriet är uppfyllt har beräknats. Som styrsignal användes sedan det u(0) som beräknats för $\tau = \tau_{opt}$.

3.2.2 ANPASSNING TILL LPEM.

Nästa steg är att formulera om problemet (3.4) till den form LPEM kräver. Ett problem är att bivillkoren i (3.4) innehåller likheter. Sådana klarar inte ellipsoidmetoden av eftersom lösningsområdet till bivillkoren då får volymen noll. Därför måste en störning läggas på likheterna. Detta gör vi genom att specificera en noggrannhet δ och ersätta alla likheter

$$a_i^t z = b_i \quad \text{med olikheterna}$$

$$b_i - \delta\cdot(a_i^t a_i)^{1/2} \leq a_i^t z \leq b_i + \delta\cdot(a_i^t a_i)^{1/2} \quad .$$

Ett annat problem är att förlustfunktionen (3.3) inte är strikt linjär. Gutman omvandlar förlustfunktionen till en linjär funktion genom att införa två hjälpvariabler för varje term i (3.3). Detta ökar dimensionen på problemet och därmed troligen också konvergenshastigheten. I avsnitt 1.6 nämnde vi att ellipsoidmetoden kan användas även då objektfunktionen är konvex. Eftersom (3.3) är konvex utnyttjar vi detta faktum. Vi behöver då beräkna en subgradient till förlustfunktionen varje gång vi hamnar in punkt som uppfyller bivillkoren. En subgradient till (3.3) i punkten y är

$$g = [\, g_1, \ . \ . \ . \ , g_r \,]^t, \quad r = \tau(n+m) \quad \text{där}$$

$$g_i = \begin{cases} \left.\dfrac{\partial \ell}{\partial z_i}\right|_{z=y} & \text{då } \dfrac{\partial \ell}{\partial z_i} \text{ existerar} \\ 0 & \text{för övrigt} \end{cases} \qquad (3.5)$$

Sedan g har beräknats för punkten z_k som uppfyller bivillkoren användes objektsnittet $g^t z \leq g^t z_k$. Efter en stunds funderande inser man att precis samma test som i

fallet med en linjär objektfunktion kan användas för (3.3), dvs övre och undre begränsningar till ℓ kan beräknas enligt avsnitt 1.4.3 . Vi lägger in en programdel i proceduren TESTOBJECT (se avsnitt 2.2.1) i LPEM som beräknar subgradienten till ℓ enligt formeln (3.5).

Problemet (3.4) kan nu skrivas om till LPEM-formen (2.1) enligt:

$$m_1 = m_2 = \tau\cdot(2n + m) \quad ,$$

$$a_i = \begin{cases} A_i^t & , \ i = 1, \ldots\ldots, \tau n \\ \\ e_{i-\tau n} & , \ i = \tau n+1, \ \ldots, \ m_1 \end{cases} \quad ,$$

A_i är rad nummer i av matrisen A i (3.4) och

e_j är enhetsvektor nr j i $R^{\tau(n+m)}$, (3.6)

$$b = [\ 0 \ldots\ldots 0 \ (\phi x(0))^t \ \tfrac{1}{2}\cdot(s + v)^t]^t \quad ,$$

b börjar med $n(\tau-1)$ st nollor. Slutligen är

$$c_i = \begin{cases} \delta\cdot(a_i^t a_i^t)^{1/2} & , \ i = 1, \ldots., \tau n \\ \\ \tfrac{1}{2}\cdot(v_{i-\tau n} - s_{i-\tau n}) & , \ i = \tau n, \ \ldots., \ m_1 \end{cases} \quad .$$

3.2.3 DUBBELINTEGRATORN.

Samplad med samplingsintervallet 1 sekund beskrivs, i tillståndsform, dubbelintegratorn av

$$x(t+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} u(t) \ , \qquad x \in R^2 \ , u \in R \ .$$

De olika vektorerna i (3.3) och (3.6) väljes till

$$r = w = c = d = [\ 1 \quad 1\ ] \ ,$$

$$\gamma_x = [\ -90 \ -2\ ]^t \quad \text{och}$$

$$\delta_x = [\ 90 \quad 2\ ]^t \ .$$

Begränsningarna på tillstånden och insignalen är

$$-100 \leq x_1(t) \leq 100 \ ,$$

$$-3 \leq x_2(t) \leq 3 \quad \text{och}$$

$$-1 \leq u(t) \leq 1 \ .$$

Initialvärdet är $x(0) = [\ -10 \quad 0\ ]^t$.

Noggrannheten δ i likheterna väljes till $\delta = 1 \cdot 10^{-3}$ och stoppkriteriet i LPEM väljes till $\varepsilon = 1 \cdot 10^{-3}$, dvs förlustfunktionen (3.3) ska beräknas med ett absolut fel $< \varepsilon$.

Startvärdena till LPEM väljes till

$$z_0 = \frac{1}{2} \cdot (s + v) \qquad \text{och}$$

$$Q_0 = \text{diag}\ [q_1, \ \ldots, q_{3\tau}\ ] \ \text{där}$$

$$q_i = \frac{1.01 \cdot (3\tau)^{1/2}}{2} \cdot (v_i - s_i) \qquad , \ i = 1, \ \ldots, 3\tau \ .$$

I tabell 3.2 visas några resultat från datorkörningar med olika τ. Samtliga beräkningar i LPEM är utförda i dubbel precision. Exekveringstiderna kan jämföras med tiderna för samma problem kört med PRIMAL, tabell 3.3. PRIMAL är en LP-rutin utvecklad vid Institutet för Tillämpad Matematik, Stockholm. PRIMAL är skrivet i FORTRAN och använder simplex-metoden med basinversen lagrad i produktform. Obsrvera att trots att PRIMAL arbetar med fler variabler än LPEM är det samma problem som löses. Anledningen till det större antalet variabler i PRIMAL är att förlustfunktionen här har omvandlats till en strikt linjär funktion genom införandet av hjälpvariabler.

Som synes är tiderna inte speciellt smickrande för LPEM. Man lägger märke till att antalet iterationer i tabell 3.1 minskar då τ ökar från 1 till 2 och från 4 till 5. Detta är en indikation pÅ att allt inte står rätt till. En närmare kontroll ger vid handen att LPEM förmår leverera en optimal lösning endast för $\tau = 1$. För alla $\tau > 1$ hittar LPEM en punkt som uppfyller bivillkoren, men då de elliptiska kloten ska konvergera mot den optimala lösningen inverkar avrundningsfelen alltför mycket; slutligen hamnar de elliptiska kloten helt och hållet utanför det tillåtna området och α blir >1, se avsnitt 1.1 . Den bästa lösningen LPEM gav innan algoritmen terminerades för $\tau = 7$ visas i tabell 3.4, där också lösningen från PRIMAL visas. Lösningen från PRIMAL är korrekt. Ur tabellen syns klart att LPEM har kommit en bit på "rätt väg" innan algoritmen terminerades.

Bland orsakerna till LPEM:s misslyckande är troligen den främsta den att det tillåtna området definierat av bivillkoren i (3.4) har en mycket liten volym samtidigt som det är mycket utsträckt i en del riktningar. Som vi förutspådde i avsnitt (1.6) tycks detta vara mycket ogynnsamt för EM. En annan orsak är att beräkninarna av parametrarna τ, π och d, (se sid 21) är mycket känsliga. Om t.ex. τ har beräknats med ett visst relativt fel så blir även det relativa felet varje koordinat i nästa iteration av samma storleksordning. Detta innebär att hela det elliptiska klotet blir translaterat från sin rätta plats. Ett liknande resonemang angående τ och δ ger vid handen att ellipsoiderna blir deformerade och vridna. Kombineras nu detta med det faktum att de elliptiska kloten är mycket "utsträckta" (cigarr- eller diskusformade) så inser man konsekvenserna lätt kan bli ödesdigra; även ett litet fel i beräkningen av x_{k+1} kan leda till att ett "cigarrformat" elliptiskt klot hamnar helt och hållet utanför det tillåtna området.

Tabell 3.2 Data från körning av dubbelintegratorn med LPEM.

Tids-horisont	Antal variabler	Antal iterationer	Exekveringstid CPU-sekunder
1	3	49	0.38
2	6	30	0.34
3	9	53	1.29
4	12	98	4.31
5	15	48	3.63
6	18	162	17.87
7	21	262	42.97

Tabell 3.3 Data från körning av dubbelintegratorn med PRIMAL.

Tids-horisont	Antal variabler	Exekveringstid CPU-sekunder
1	6	0.05
2	15	0.13
3	24	1.18
4	33	0.25
5	42	0.31
6	51	0.38
7	60	0.48

Tabell 3.4 LPEM- och PRIMAL-lösning för $\tau = 7$. PRIMAL-lösningen är korrekt.

	LPEM-lösning	PRIMAL-lösning
$u(0) =$	0.749	1.000
$x_1(1) =$	-9.626	-9.500
$x_2(1) =$	0.749	1.000
$u(1) =$	0.878	1.000
$x_1(2) =$	-8.438	-8.000
$x_2(2) =$	1.627	2.000
$u(2) =$	0.036	0.000
$x_1(3) =$	-6.793	-6.000
$x_2(3) =$	1.662	2.000
$u(3) =$	0.139	0.000
$x_1(4) =$	-5.062	-4.000
$x_2(4) =$	1.801	2.000
$u(4) =$	-0.217	0.000
$x_1(5) =$	-3.369	-2.000
$x_2(5) =$	1.584	2.000
$u(5) =$	0.062	-1.000
$x_1(6) =$	-1.755	-0.500
$x_2(6) =$	1.645	1.000
$u(6) =$	-0.725	-1.000
$x_1(7) =$	-0.472	0.000
$x_2(7) =$	0.920	0.000

Observera att problemet (3.4) kan formuleras om till ett problem i endast $\tau \cdot m$ variabler med likhetsvillkoren eliminerade (se avsnitt 1.7). Applicerar vi LPEM på detta reducerade problem torde resultatet bli betydligt gynnsammare.

4. SAMMANFATTNING MED SLUTSATSER.

Den här uppsatsen behandlar ellipsoidmetoden för linjär programmering. Metoden beskrivs både ur teoretisk och praktisk synpunkt. Ett program för metoden presenteras och testas på två exempel, Klee-Minty's problem och dubbelintegratorn styrd med LP-regulatorn. Programmet är skrivet för att vara lätt att modifiera och är således inte optimerat för att vara så snabbt som möjligt.

Resultaten från datorkörningarna är blandade. Körningarna på Klee-Minty's problem antyder att antalet iterationer som krävs är begränsat av ett polynom i antalet variabler medan körningarna på dubbelintegratorn visar att numeriska problem kan uppstå.

De främsta nackdelarna med metoden tycks vara:

1) Metoden kräver mycket minnesutrymme. En n*n-matris ska lagras och uppdateras i varje iteration.

2) Problem med den numeriska stabiliteten kan uppstå då de elliptiska kloten är mycket "ovala".

3) Konvergenshastigheten beror i hög grad på valet av startellipsoid. Det är viktigt att välja en startellipsoid med så liten volym som möjligt, som samtidigt innehåller lösningspunkten.

Förhållandet i punkt ett kan förbättras genom att använda sig av L^tDL-faktorisering. Det krävda minnesutrymmet kommer då i det närmaste att halveras, varvid även arbetet med uppdateringen minskar avsevärt.

Problemet i 2) kan troligen minskas ned genom att succesivt sträva efter att välja snitt som är så nära vinkelräta som möjligt mot snitten i de närmast föregående iterationerna.

Bland de många olika metoderna att implementera ellipsoidmetoden som är kända är troligen den mest effektiva den som använder sig av L^tDL-faktorisering och metoden med objektsnitt. Helt klart är dock att mycket arbete återstår innan man med ellipsoidmetoden kan uppnå prestanda jämförbara med simplexmetodens.

5. REFERENSER.

Direkt använda referenser:

[1] Berresford G.C, Rocket A.M och Stevenson J.C: Khachiyan's Algorithm, Part 1 och Part 2. Två artiklar införda i BYTE augusti 1980, 198-208 och september 1980, 242-255.

[2] Bland R.G, Goldfarb D. och Todd M.J: The Ellipsoid Method: A Survey. Operations Research, vol 29, No 6, 1039-1091, 1980.

[3] Carleson L: Hacian's method in linear programming. Report No 3, 1980. Inst. för Matematik, Lunds Universitet.

[4] Goldfarb D., Todd M.J: Modifications And Implementation of The Shor-Khachian Algorithm For Linear Programming. Department of Computer Science, Cornell University, Ithaca, New York 14853.

[5] Gutman P-O: Application of linear programming for on-line control. Del 3 i "Controllers for bilinear and constrained linear systems", teknisk doktorsavhandling, 1982, Dept. Auto. Contr., Lund Inst. of Techn., Lund, Sweden.

[6] Gårding L: 2.5 Ellipsoider och 7.6 Ellipsoidmetoden. Manuskript, 1982, Inst. för Matematik, Lunds Universitet.

[7] Khachijan L.G: A Polynomial Algorithm in Linear Programming. Doklady Akademiia Nauk SSSR 244, 1093-1096, 1979 (översatt till engelska i Soviet Mathematics Doklady v20, 191-194, 1979).

Almänna referenser:

Eriksson G: Numerisk Analys Fortsättningskurs. Inst. för Informationsbehandling, Lund 1979, Sigma-tryck.

Holmberg G: PRIM2- ett subrutinpaket som löser LP-problem. UTM-Program nr 50, 1981. Utdragsgruppen för tillämpad matematik. Kungliga Tekniska Högskolan, Stockholm.

Jones C.P, Marwil E.S: A Dimensional Reduction Variant of the Ellipsoid Algorithm for Linear Programming Problems. EG & G Idaho, Inc. Idaho Falls, Idaho, 1980.

Lova'sz L: A New Linear Programming Algorithm - Better or Worse Than the Simplex Method? Math. Intelligencer 2, 141-146, 1980.

Padberg M.N , Rao M.R: The Russian Method for Linear Inequalities, Graduate School of Administration, New York University, New York, 1980.

Aström K.J: Reglerteori. Stockholm: Almqvist & Wiksell, 1976.

Aström K.J , Wittenmark B: Computer Control Theory, Dept. Auto. Contr., Lund Inst. of Techn., Lund, Sweden, 1981.

APPENDIX A. BRUKSANVISNING FÖR LPEM.

LPEM är ett underprogram som löser linjärprogrammerings-problem. Programmet är skrivet i PASCAL.

Användaren av LPEM ska själv ombesörja

1) inläsning och transformation av problemet till den form som LPEM kräver.

2) dimensionering och deklaration av i rutinen ingående fält.

3) initiering av vissa variabler och fält.

1. PROBLEMFORMULERING.

LPEM kräver att problemet formuleras enligt:

$$\text{minimera } a_0^t x \ , \ x \in R^n \text{ då}$$

$$b_i - c_i \leq a_i^t x \leq b_i + c_i \ , \ c_i > 0 \ , \ i = 1, \ldots, m_1 \quad \text{och}$$

$$a_i^t x \leq b_i \ , \ i = m_1 + 1, \ldots, m_2 \ .$$

2. ANROP.

LPEM anropas (från PASCAL) med satsen

```
LPEM(UTFIL,X,Y,R,OMEGA,QROW,Q,JD,ACJD,C,B,A,AROW,
     ACOL,N,M1,M2,OL,RES,EPSILON,OBJ,STANDARD);
```

Samtliga argument ska deklareras av användaren. Hur detta ska ske beskrivs i de följande avsnitten.

3. TYPDEKLARATIONER.

Inför följande beteckningar:

EL = sammanlagda antalet element skilda från noll i vektorerna a_i

N = antal variabler

M2 = antalet vektorer a_i

$S = (N^2+3N-2)/2$

Följande typer ska deklareras.

```
VECTOR        = ARRAY [1..N] OF REAL;
ELLIPSOIDMAT  = ARRAY [1..N] OF VECTOR;
ORTMATRIX     = ARRAY [1..S] OF REAL;
ACCESSORT     = ARRAY [1..N] OF INTEGER;
RIGHTHANDSIDE = ARRAY [1..M2] OF REAL;
LEFTHANDSIDE  = ARRAY [1..EL] OF REAL;
ACCESSROW     = ARRAY [0..(M2+1)] OF INTEGER;
ACCESSCOL     = ARRAY [1..EL] OF INTEGER;
```

4. DEKLARATIONER

Invariablerna ska vara deklarerade enligt följande:

```
UTFIL              : TEXT;
X,Y,R,OMEGA,QROW   : VECTOR;
Q                  : ELLIPSOIDMAT;
JD                 : ORTMATRIX;
ACJD               : ACCESSORT;
C,B                : RIGHTHANDSIDE;
A                  : LEFTHANDSIDE;
AROW               : ACCESSROW;
ACOL               : ACCESSCOL;
N,M1,M2,OL,RES     : INTEGER;
OBJ,EPSILON        : REAL;
STANDARD           : BOOLEAN;
```

5. LAGRING AV PROBLEMET

LPEM kräver att problemet lagras enligt följande:

A : RIGHTHANDSIDE.

A innehåller vektorerna a_i ; i=0,1,....,M2

lagrade vektor för vektor. Endast element skilda från noll lagras

AROW : ACCESSROW.

AROW[i] = heltal som talar om på vilken plats i A som vektor a_i börjar.

OBS! LPEM kräver att AROW[M2+1]= antal element i A +1.

ACOL : ACCESSCOL.

ACOL[j] = variabelnummer hörande till A[j]

B : RIGHTHANDSIDE.

B[i]= b_i

C : RIGHTHANDSIDE.

C[i]= c_i

N,M1,M2 : INTEGER.

N = antal variabler

M1 = antal parallella par av olikheter (=m_1)

M2 = totala antalet vektorer a_i-1 (=m_2)

6. REPRESENTATION AV LÖSNINGEN.

X[I] : VECTOR.

X[I] = värdet på variabel nummer I

OBJ : REAL.

OBJ = ojektfunktionens värde

RES : INTEGER.

RES = 1 Lösningen är optimal.

RES = 2 Ingen lösning existerar*.

RES = 3 Lösningen är tillåten men inte säkert optimal.

RES = 4 Matrisen Q har tappat rang*. Aktuell lösning är tillåten.

RES = 5 Matrisen Q har tappat rang*. Aktuell lösning är inte tillåten.

*) Se del 8. INITIERINGAR.

7. UTSKRIFT.

UTFIL : TEXT.

UTFIL är namnet på den fil på vilken utskrift önskas.

OL : INTEGER.

OL = önskad utskriftsnivå (0,1,2). 0 ger ingen uskrift, 2 ger mycket omfattande utskrifter.

STANDARD : BOOLEAN.

STANDARD = TRUE om utfile är standardfilen OUTPUT.

STANDARD = FALSE annars.

8. INITIERINGAR.

X : VECTOR, Q : ELLIPSOID.

X och Q ska innan anropet tilldelas sådana värden, x_0 och Q_0, att problemets lösning, om sådan existerar, är innehållet i området

$$\{x \in R^n \mid x = x_0 + Q_0 z,\ |z| \leq 1\}.$$

Om den optimala lösningen inte finns i detta område kommer LPEM inte att hitta den.

EPSILON : REAL.

EPSILON ska tilldelas ett värde > 0. Den exakta lösningen kommer att ligga i intervallet [OBJ-EPSILON,OBJ]. Observera att matrisen Q kan tappa rang. I så fall kan LPEM inte forsätta itererandet utan termineras. För att undvika detta bör EPSILON inte väljas alltför litet. EPSILON≥0.5E-6 rekommenderas.

OL : INTEGER, STANDARD : BOOLEAN.

Se avsnitt 7. UTSKRIFT.

APPENDIX B. PROGRAMLISTA AV LPEM.

```
procedure LPEM(var OUTFILE:TEXT;var X,Y,R,OMEGA,QROW:VECTOR;
               var Q:ELLIPSOIDMAT;var JD:ORTMATRIX;var ACJD:
               ACCESSJD;var EPS,B:RIGHTHANDSIDE;var A:
               LEFTHANDSIDE;var AROW:ACCESSR;var ACOL:
               ACCESSC;var N,M1,M2,OL,RES:INTEGER;
               var OBJ,EPSILON:REAL; STANDARD:BOOLEAN);

var ITER,NR                      : INTEGER;
    READY,FEAS,OPT               : BOOLEAN;
    NO,N1,N2,CPUTIME,CPUST       : REAL;
    OBJMIN,OBJMAX                : REAL;
    TAU,DELTA,PI                 : REAL;

    procedure INITIATE;

    var I,J  :INTEGER;

    begin
        { starta klockan }
        CPUST:=CLOCK;
        if not STANDARD then
            rewrite(OUTFILE);
        { accessvektor till JD }
        ACJD[1]:=0;
        for I:=2 to N do
            ACJD[I]:=(I-1)*N-(I-2)*(I-3) div 2;
        NO:=1/(N+1);
        N1:=N/sqrt(sqr(N)-1);
        { faktorn N2 ska kompensera för avrundningsfel }
        N2:=exp(ln(2)/(8*sqr(N)));
        OBJMAX:=1E+20;
        OBJMIN:=-1E+20;
        ITER:=0;
        READY:=false;
        FEAS:=false;
        FIRSTFEAS:=true;
        OPT:=false;
        NR:=0;
    end;
    { end INITIATE }

    procedure CUTELLIPSOID;

    var SIGN                                  : -1..1;
        F,F1,GAMMA,RO,SIGMA                   : REAL;
        ALFA,ALFA1,SQAL,SQAL1                 : REAL;
        FOUND,PARALLELCUT,OBJECTCUT           : BOOLEAN;
        I,J,K,START,STOP                      : INTEGER;
```

```
procedure NEWOMEGA;

var I,J       : INTEGER;
    DI        : REAL;

begin
    GAMMA:=0;
    for I:=1 to N do
    begin
        DI:=0;
        for J:= START to STOP do
            DI:=DI+Q[ACOL[J],I]*A[J];
        OMEGA[I]:=DI;
        GAMMA:=GAMMA+sqr(DI);
    end;
    GAMMA:=sqrt(GAMMA);
    { om GAMMA=0 itererandet inte fortsätta
      eftersom 1/GAMMA behövs senare }
    if GAMMA=0 then
    begin
        READY:=true;
        if OL)=1 then
        begin
            write(OUTFILE,'Q har tappat rang.   ');
            writeln(OUTFILE,'GAMMA=',GAMMA:10);
       end;
    end
    else
    begin
        for I:=1 to N do
            OMEGA[I]:=OMEGA[I]/GAMMA;
    end;
end;
{ end NEWOMEGA }


procedure SELECTACUT;

    procedure TESTPARALLELCONSTRAINT;

    begin
        if abs(F))EPS[NR] then
        begin
            { ett av ≤- eller ≥-villkoret
              är inte uppfyllt }
            FOUND:= true;
            PARALLELCUT:=true;
            OBJECTCUT:=false;
            if F)EPS[NR] then
            begin
                F1:=-(EPS[NR]+F);
                F:=F-EPS[NR];
                SIGN:=1;
```

```
        end
        else
        begin
            { ≥-villkoret är inte uppfyllt.
              Tecknet korrigeras genom SIGN:=-1 }
            F1:=F-EPS[NR];
            F:=-(F+EPS[NR]);
            SIGN:=-1;
        end;
    end;
end;
{ end TESTPARALLELCONSTRAINT }


procedure TESTCONSTRAINT;

begin
    if F>0 then
    begin
        FOUND:= true;
        PARALLELCUT:=false;
        OBJECTCUT:=false;
        SIGN:=1;
    end;
end;
{ end TESTCONSTRAINT }


procedure TESTOBJECT;

var OBJM : REAL;
    I    : INTEGER;

begin
    OBJ:=0;
    START:=AROW[0];
    STOP:=AROW[1]-1;
    { beräkna objekets värde för det
      aktuella X-värdet }
    for I:=START to STOP do
        OBJ:=OBJ+A[I]*X[ACOL[I]];
    if OBJ<OBJMAX then
    begin
        { Den bästa punkten hittills. Spara
          undan och uppdatera OBJMAX }
        OBJMAX:=OBJ;
        Y:=X
    end;
    { Uppdatera OBJMIN. GAMMA beräknas i NEWOMEGA }
    NEWOMEGA;
    OBJM:=OBJ-GAMMA;
    if OBJM>OBJMIN then
        OBJMIN:=OBJM;
    if OBJMAX-OBJMIN<EPSILON then
    begin
```

```
                    { Minimum har beräknats med
                      noggrannheten EPSILON. }
                    READY:=true;
                    OPT:=true;
                    X:=Y;
                    OBJ:=OBJMAX;
                end;
                PARALLELCUT:=false;
                OBJECTCUT:=true;
                SIGN:=1;
                F:=0;
            end;
            { end TESTOBJECT }


        begin
            FOUND:= false;
            ITER:=ITER+1;
            J:=0;
            while (J<M2) and not FOUND do
            begin
                { I denna loop beräknas ett villkor och testas.
                  Loopen lämnas så snart ett villkor inte
                  är uppfyllt (FOUND sätts till true). }
                J:=J+1;
                NR:=NR+1;
                if NR=M2+1 then
                    NR:=1;
                START:=AROW[NR];
                STOP:=AROW[NR+1]-1;
                F:=-B[NR];
                for I:=START to STOP do
                begin
                    K:=ACOL[I];
                    F:=F+A[I]*X[K];
                end;
                if NR<=M1 then
                    TESTPARALLELCONSTRAINT
                else
                    TESTLESSTHAN;
            end;
            if not FOUND then
            begin
                TESTOBJECT;
                FEAS:=true;
            end;
        end;
        { end SELECTACUT }

    begin
        SELECTACUT;
        if not READY then
        begin
            if not OBJECTCUT then
                NEWOMEGA;
```

```
        ALFA:=F/GAMMA;
        if ALFA)1 then
            READY:= true;
    end;
    if not READY then
    begin
        if PARALLELCUT then
        begin
            ALFA1:=F1/GAMMA;
            if -ALFA1(1 then
            begin
                SQAL:=sqr(ALFA);
                SQAL1:=sqr(ALFA1);
                RO:=4*(1-SQAL)*(1-SQAL1);
                RO:=sqrt(RO+sqr(N*(SQAL-SQAL1)));
                SIGMA:=2*(1-ALFA*ALFA1-RO/2)/sqr(ALFA-ALFA1);
                SIGMA:=NO*(N+SIGMA);
                { SIGMA kan pga avrundningsfel bli )1. I så
                  fall kan parallellt snitt inte göras
                  eftersom sqrt(1-SIGMA) behövs senare. }
                if SIGMA)=1 then
                    PARALLELCUT:=false;
            end
            else
                PARALLELCUT:=false;
        end;
        if PARALLELCUT then
        begin
            DELTA:=1-(SQAL+SQAL1-RO/N)/2;
            DELTA:=N1*N2*sqrt(DELTA);
            PI:=DELTA*sqrt(1-SIGMA);
            TAU:=(ALFA-ALFA1)*SIGMA*SIGN/2;
        end
        else
        begin
            DELTA:=N1*N2*sqrt(1-sqr(ALFA));
            PI:=N*NO*N2*(1-ALFA);
            TAU:=NO*(1+N*ALFA)*SIGN;
        end;
    end;
    if OL=2 then
    begin
        writeln(OUTFILE,'ITER=',ITER:6);
        if FOUND then
        begin
            write(OUTFILE,'FOUND=',FOUND,'; NR=',NR:5);
            write(OUTFILE,'; F=',F:8:3);
            writeln(OUTFILE,'; PARALLELCUT',PARALLELCUT);
        end;
        if OBJECTCUT then
        begin
            write(OUTFILE,'OBJECTCUT=',OBJECTCUT);
            write(OUTFILE,'; OBJMIN=',OBJMIN:8:5);
            writeln(OUTFILE,'; OBJMAX=',OBJMAX:8:5);
        end;
```

```
            writeln(OUTFILE,'ALFA=',ALFA:8:7);
            if PARALLELCUT then
            begin
                write(OUTFILE,'ALFA1=',ALFA1:8:7);
                writeln(OUTFILE,'; SIGMA=',SIGMA:8:7);
            end;
            write(OUTFILE,'TAU=',TAU:12,'; PI=',PI:12);
            writeln(OUTFILE,'; DELTA=',DELTA:12);
        end;
        if READY then
        begin
            if OPT then
                RES:=1
            else if FEAS then
            begin
                if GAMMA=0 then
                    RES:=4
                else
                    RES:=3
            end
            else if GAMMA=0 then
                RES:=5
            else
                RES:=2
        end;
        if READY and (OL)=1) then
        begin
            writeln(OUTFILE);
            writeln(OUTFILE,'READY=',READY,'; RES=',RES:3);
            writeln(OUTFILE,'OBJ=',OBJ:12);
            writeln(OUTFILE,'X=');
            for I:=1 to N do
            begin
                write(OUTFILE,X[I]:12);
                if I mod 10=0 then writeln(OUTFILE);
            end;
            writeln(OUTFILE);
            writeln(OUTFILE,'ITER=',ITER:6);
            writeln(OUTFILE,'ALFAMEDEL=',ALFASUM/ITER:12);
            CPUTIME:=(CLOCK-CPUST)*0.001;
            writeln(OUTFILE,'CPUTIME=',CPUTIME:9:4,' sec.');
        end;
    end;
    { end CUTELLIPSOID }


    procedure NEWELLIPSOID;

        procedure NEWX;

        var I,J    : INTEGER;
            X1     : REAL;

        begin
            for I:=1 to N do
```

```
    begin
        X1:=0;
        for J:=1 to N do
            X1:=X1+Q[I,J]*OMEGA[J];
        X[I]:=X[I]-TAU*X1;
    end;
    if OL=2 then
    begin
        writeln(OUTFILE,'X=');
        for I:=1 to N do
        begin
            write(OUTFILE,X[I]:13);
            if I mod 9=0 then writeln(OUTFILE);
        end;
        writeln(OUTFILE);
    end;
end;
{ end NEWX }


procedure NEWQ;

var I,J,K,L,M   : INTEGER;
    R1,QIJ      : REAL;

begin
    { Beräkna först JD }
    I:=N;
    R[N]:=sqr(OMEGA[N]);
    for J:=N-1 downto 1 do
    begin
        R[J]:=R[J+1]+sqr(OMEGA[J]);
        if R[J+1]=0 then
            I:=J;
        R[J+1]:=sqrt(R[J+1]);
    end;
    R[1]:=sqrt(R[1]);
    R1:=R[1];
    for J:=1 to N do
        JD[J]:=OMEGA[J]*PI/R1;
    for J:=2 to I do
    begin
        K:=ACJD[J];
        R1:=-OMEGA[J-1]*DELTA/(R[J-1]*R[J]);
        JD[K+1]:=R[J]*DELTA/R[J-1];
        for L:=2 to N do
            JD[K+L]:=R1*OMEGA[J+L-2];
    end;
    for J:=I+1 to N do
    begin
        K:=ACJD[J];
        JD[K+1]:=0;
        JD[K+2]:=DELTA;
        for L:=3 to N-J+2 do
            JD[K+L]:=0;
```

```
                end;
                { JD är nu beräknad }

                { Beräkna ett nytt Q. Q:=QJD }
                for I:=1 to N do
                begin
                    for J:=1 to N do
                    begin
                        K:=ACJD[J];
                        if J>2 then
                            M:=J-2
                        else
                            M:=0;
                        QIJ:=0;
                        for L:=1 to N-M do
                            QIJ:=QIJ+Q[I,L+M]*JD[K+L];
                        QROW[J]:=QIJ;
                    end;
                    for J:=1 to N do
                    begin
                        QIJ:=QROW[J];
                        Q[I,J]:=QIJ;
                    end;
                end;

                if OL=2 then
                begin
                    CPUTIME:=(CLOCK-CPUST)*0.001;
                    writeln(OUTFILE,'CPUTIME=',CPUTIME:9:4,' sec.');
                    writeln(OUTFILE);
                end;
            end;
            { end NEWQ }

        begin
            NEWX;
            NEWQ;
        end;
        { end NEWELLIPSOID }


    begin
        INITIATE;
        CUTELLIPSOID;
        while not READY do
        begin
            NEWELLIPSOID;
            CUTELLIPSOID;
        end;
    end;

    { end  LPEM  }
```