UNIVERSITETET I OSLO
Matematisk institutt

MasterOppgave i matematikk

Lineær optimering i et algoritmisk perspektiv.

Aizan Magomadova

18. desember 2009

Forord

Denne hovedoppgaven er skrevet som en del av master graden i matematikk under LAP programmet ved Institutt for Matematikk, ved Universitetet i Oslo. Oppgaven krever ikke avanserte forkunnskaper i matematikk, men det er fordel med noen basiskunnskaper i lineær programmering, linear algebra og kalkulus. Ellers har jeg prøvd å presentere materialet så mye forståelig som mulig.
Jeg vil takke min veileder, Geir Dahl, for råd og tips gjennom hele arbeidet med oppgaven. Han har vært en verdifull ressurs både i korrigeringen av feil og utformingen av oppgaven. En stor takk rettes også til familievennen, Anders Brattvik, for korrekturlesing av oppgaven.
Til slutt vil jeg takke min mann og kjære barn for tålmodighet, og sist, men ikke minst marsvinen Marsipan for at hun holdt meg med selskap i løpet av lange netter.

Røyse, desember 2009,
Aizan Magomadova

Innhold

1 Innledning

Hovedmålet med oppgaven er å studere og redegjøre for *lineær optimering* i et algoritmisk perspektiv. Lineær optimering, også kalt lineær programmering (LP), er et viktig matematisk område med en rekke anvendelser. LP er forankret matematisk i lineær algebra og konveksitet/matematisk analyse. I denne oppgaven skal jeg redegjøre for to hovedtyper av algoritmer for LP samt se på en spesiell anvendelse innenfor approksimasjon av matriser.
I kapittel 2 gis en kort introduksjon til Lineær programmering, konveksitet og noen viktige definisjoner som vi får bruk for i denne oppgaven. I kapittel 3 redegjøres for simpleksalgoritmen. Kapittelet begynner med en geometri bak simpleksalgoritmen. Videre gis en matematisk beskrivelse av LP, drøftes numeriske utfordringer og effektiviteten. Det er gitt beskrivelsen av simpleksalgoritmen i matriseform og en iterasjon i algoritmen. Kapittel 4 redegjør for en *indrepunktsmetode* for LP, den så kalte *path-following method*, som kalles videre *PF-metoden*. Det er gitt en kort beskrivelse av det teoretiske grunnlaget for indrepunktsmetoder: Newton's metode for løsning av ikkelineære likninger, Lagrange's metode for optimering med lineære begrensninger og Fiacco og McCormick's barriere metode for optimering med begrensninger som er ulikheter. Det er laget tre implementasjoner av PF-metoden i Matlab. Disse algoritmene testet på en del testproblemer, og testresultatene presentert i kapittel 5. Videre sammenliknet algoritmene med simpleksalgoritmen. I kapittel 6 ser vi på en spesiell anvendelse innenfor matriseapproksimasjon. Det studeres et optimeringsproblem, hvor objektivfunksjonen er en avstandfunksjon $d(A, B)$ for to matriser $A, B \in \mathbb{R}^{m\times n}$. Det er testet PF-algoritmen på et problem av denne typen. Videre brukes rapporten av Boyd og Dattorro til å foreslå og analysere en alternativ metode for problemet ved hjelp av alternerende projeksjoner.

2 Introduksjon til LP

Lineær programmering - en matematisk disiplin viet til teorien og metodene for å løse problemer på lineære funksjoner i n-dimensjonalt vektorrom definert av systemer av lineære likninger og ulikheter.

Lineær programmering (LP) er et spesielt tilfelle av konveks programmering, som igjen er et spesielt tilfelle av matematisk programmering. Samtidig er det grunnlaget for flere metoder for å løse problemene med heltall og ikke-lineær programmering.

Mange av egenskapene til lineære programmeringsproblemer kan tolkes som egenskaper av polyhedere og dermed geometrisk formulere og bevise dem.

Begrepet "programmeringen" må forstås i betydningen "planlegging". Det ble foreslått i midten av 1940 av George Danzig, en av grunnleggerne av lineær programmering, selv før datamaskiner ble brukt til å løse lineære optimeringsproblemer.

2.1 Matematisk formulering av LP problem

Lineær optimering (lin.opt.= lin.programmering) er å maksimere en lineær funksjon i flere variable på formen

$$f(x) = \sum_{j=1}^{n} c_j x_j = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \tag{2.1}$$

med begrensninger

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad \texttt{ved} \qquad i = 1, 2, \ldots, m$$

Disse begrensningene kan bestå av enten likninger eller ulikheter som er en lineær kombinasjon av beslutningsvariabler [Vand08]:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b. \tag{2.2}$$

Funksjonen $f(x)$ kalles *objektivfunksjonen*. Et forslag til verdier for beslutningsvariablene kalles *en løsning*. En løsning kalles *tillatt* hvis den tilfredsstiller alle begrensningene. En vektor $x^* = (x_1, x_2, \ldots, x_n)$ kalles en *optimal løsning* av problemet (2.1) hvis den har den største objektivverdi blant alle vektorer som tilfredsstiller alle begrensningene.
Det er lett å konvertere begrensningene fra en form til en annen. F.eks., en begrensning i form av en ulikhet

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \leq b \tag{2.3}$$

kan konverteres til en likning ved å addere en ikke-negativ variabel w, som kalles en slakkvariabel:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + w = b, \qquad w \geq 0. \tag{2.4}$$

I tillegg kan en likning

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b \tag{2.5}$$

erstattes av to ulikheter

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b \tag{2.6}$$

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b \tag{2.7}$$

Fra et matematisk synspunkt er dette en foretrukket presentasjon. Ut fra dette kan lineær programmering problem formuleres på følgende måte:

$$\begin{array}{rrll} \text{maksimer} & c_1x_1 + c_2x_2 + & \ldots & +c_nx_n \\ \text{f.at} & & & \\ & a_{11}x_1 + a_{12}x_2 + & \ldots & +a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + & \ldots & +a_{2n}x_n \leq b_2 \\ & & \vdots & \\ & a_{m1}x_1 + a_{m2}x_2 + & \ldots & +a_{mn}x_n \leq b_m \\ & x_1, x_2, & \ldots & x_n \geq 0. \end{array}$$

Vi skal henvise til lineære programmer formulert på denne måten som lineære programmer i *standard form*. Vi skal alltid bruke m til å betegne antall begrensninger og n til å betegne antall beslutningsvariabler.
Senere skal vi jobbe med LP problemer på matriseform som ser slik ut

$$\begin{array}{rl} \text{maksimer} & c^Tx \\ \text{f.at} & Ax \leq b \\ & x \geq 0. \end{array}$$

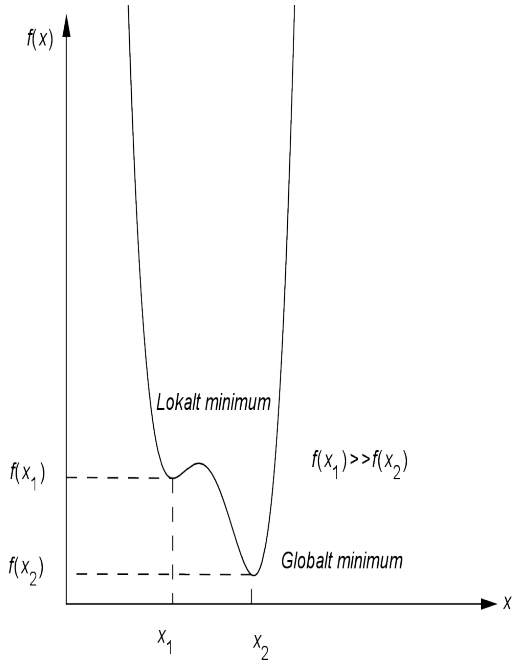
Her er x og c kolonnevektorer med n komponenter. A er en $m \times n$ matrise og b er en kolonnevektor med m komponenter. O står for nullvektoren av en passende lengde.
Optimeringsproblemet kalles *lineært* hvis objektivfunksjonen f og begrensningene er lineære, dvs., tilfredsstiller

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{2.8}$$

for alle x og $y \in \mathbb{R}^n$ og $\alpha, \beta \in \mathbb{R}$. Hvis optimeringsproblemet ikke er lineært, så kalles det et *ikke-lineært* program.

2.2 Lineær optimering og konveksitet.

Et optimeringsproblem er å minimere en funksjon f i flere variabler, og det ideelle vil være å finne et punkt x^* slik at $f(x^*) \leq f(x)$ holder for alle andre punkter x fra D_f. En slik løsning x^* kalles en *global optimal løsning* [Dahl04]. Siden det er vanskelig å finne et globalt minimum, nøyer man seg med et lokalt munimumspunkt[Dahl08] som oppfyller $f(x^*) \leq f(x)$ for $\forall x$ i D_f tilstrekkelig nær x^*. Det er flere optimerings algoritmer som kan brukes til å finne lokale minimum av f. Problemet er at en funksjonsverdi i et lokalt minimum hos en funksjon kan være mye større enn funksjonsverdi i et globalt minimum [Dahl08].



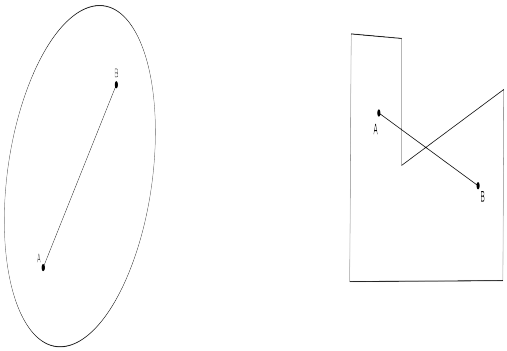
Figur 1: Funksjonsverdi i et lokalt minimum er mye større enn i et globalt minimum.

Det er naturlig med spørsmål: finnes det funksjoner som har lokalt minimum som er globalt minimum? I følge [Dahl08]:
Hvis f er en konveks funksjon, og D_f er en konveks mengde, så er et lokalt minimumspunkt et globalt minimumspunkt.
Fra Kalkulus (se [Lind06]) vet vi at hvis en funksjon $f(x)$ definert over et intervall, har kontinuerlig derivert $f'(x)$ og oppnår minimum (eller maksimum) i x_0 fra dette intervallet, så $f'(x_0) = 0$. Men først noen definisjoner:

Definisjon 2.1. *Hvis $x, y, p \in \mathbb{R}^n$ og $p = \lambda x + (1-\lambda)y$, $0 \leq \lambda \leq 1$, da er p en konveks kombinasjon av x og y. Generelt, p er en konveks kombinasjon av punkter x_1, $x_2, \ldots, x_r$ hvis $p = \sum_{i=1}^{r} \lambda_i x_i$, $\lambda_i \geq 0$, og $\sum_{i=1}^{r} \lambda_i = 1$. Punkt p er en streng konveks kombinasjon av punkter $x_1, x_2, \ldots, x_r$ hvis p er en konveks kombinasjon av $x_1, x_2, \ldots, x_r$ og $p \neq x_i$ for alle i.*

Definisjon 2.2. *$C \subseteq \mathbb{R}^n$ er en konveks mengde hvis for alle x og $y \in C$, har vi $\lambda x + (1-\lambda)y \in C$ for alle $\lambda \in [0,1]$. Da ligger linjestykke mellom x og y i C.*

Kombinasjon av konvekse mengder er konveks, og en konveks kombinasjon av punkter i en konveks mengde er i denne mengden.
Gitt en mengde $B(a,r) = \{x \in \mathbb{R}^n : \|x - a\| < r\}$ som består av de punktene i $\mathbb{R}^n$ som har avstand mindre enn r til punktet a. Vi kaller $B(a,r)$ *en (åpen) kule om a med radius r*. En mengde $C \subseteq \mathbb{R}^n$ kalles *åpen* hvis den inneholder en (åpen) kule rundt hvert av sine punkter, dvs. for hver $x \in C$ fins en $\epsilon > 0$ slik at $B(a,\epsilon) \subseteq C$. En mengde $C \subseteq \mathbb{R}^n$ kalles *lukket* hvis komplementet $\bar{C}=$



Figur 2: Konveks og ikke konveks (konkav) mengde.

$\{x \in \mathbb{R}^n : x \notin C\}$ er en åpen mengde.
En mengde C i $\mathbb{R}^n$ kalles *begrenset*, hvis det fins et tall M slik at $\|x\| \leq M$ for alle $x \in C$. En lukket og begrenset mengde er *kompakt*, og i optimeringsproblemer er tillattmengde nesten alltid lukket, og ofte kompakt.

Definisjon 2.3. *Betrakt en funksjon $f : C \to \mathbb{R}$, hvor $C \subseteq \mathbb{R}^n$ er en konveks mengde. Vi kaller f konveks hvis for alle x og $y \in \mathbb{R}^n$ og alle $0 \leq \lambda \leq 1$ holder ulikheten*

$$f\left((1-\lambda)x + \lambda y\right) \leq (1-\lambda)f(x) + \lambda f(y). \tag{2.9}$$

Hvis ulikheten er streng, dvs. $0 < \lambda < 1$ og $x \neq y$, så er f *streng konveks.*

Teorem 2.4. *Funksjonen f er konveks hvis og bare hvis H(x) er psd*[1] *for $\forall x \in D_f$. Hvis $H(x)$ er pd for $\forall x \in D_f$, så er f streng konveks.*

La nå $f : \mathbb{R}^n \to \mathbb{R}$ være en konveks funksjon. Betrakt et optimeringsproblem hvor vi minimerer f over $\mathbb{R}^n$. Vi kaller $x_0 \in \mathbb{R}^n$ et *globalt minimum* for optimeringsproblemet hvis $f(x_0) \leq f(x)$ for alle $x \in \mathbb{R}^n$.
Hvis ulikheten holder for alle x tilstrekkelig nær x_0, så har vi et *lokalt minimum.*
Funksjonen f definert på en åpen mengde med verdier i $\mathbb{R}^n$ kaller vi *deriverbar* i punktet x_0, hvis det fins en vektor d slik at

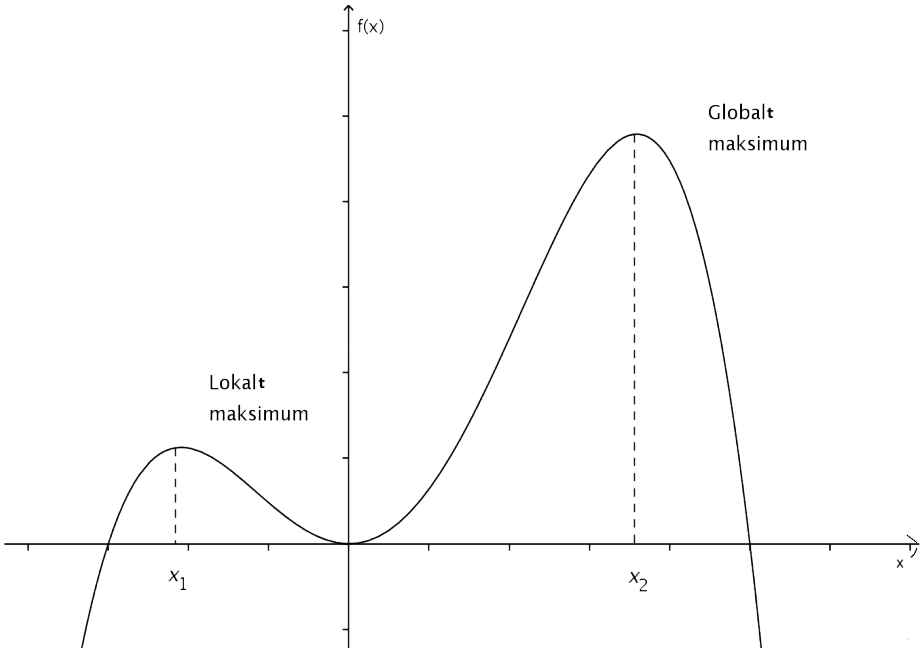
$$\lim_{h \to O} (f(x_0 + h) - f(x_0) - d^T h)/\|h\| = 0.$$

Denne vektoren d er så viktig at den har fått sitt eget navn og sitt eget symbol. Anta at de partiellderiverte til f eksisterer i et punkt $x_0 \in \mathbb{R}^n$. Da kalles

$$\nabla f(x_0)^T = \left(\frac{\partial f}{\partial x_1}(x_0), \frac{\partial f}{\partial x_2}(x_0), \ldots, \frac{\partial f}{\partial x_n}(x_0)\right)$$

gradienten til f i punktet x_0.

[1] En lineær operator $S : \mathbb{R}^n \to \mathbb{R}^n$ kalles *positiv semidefinite* (psd) hvis S er selvadjungert og $\langle x, Sx\rangle \geq 0$ for alle $x \in \mathbb{R}^n$. Hvis $\langle x, Sx\rangle > 0$ for alle $x \neq 0$, så kalles S *positiv definite* (pd). En lineær operator $S : \mathbb{R}^n \to \mathbb{R}^n$ kalles *selvadjungert* hvis $S = S^*$. Funksjonen f er *dobbelt deriverbar* i $x \in D_f$ hvis f *kontinuerlig deriverbar* og $\exists$ lineær operator $H(x) : \mathbb{R}^n \to \mathbb{R}^n$ som oppfyller $\lim_{\|\delta x\| \to 0} \dfrac{\|g(x+\delta x) - g(x) - H(x)\|}{\|\delta x\|} = 0$. Hvis den eksisterer, så kalles $H(x)$ *Hessian* av f. $H(x)$ på matriseform har $\dfrac{\partial^2 f}{\partial x_j \partial x_i}$ som koeffisienter.



Figur 3: Lokalt og globalt maksimum.

Teorem 2.5. *La $f\colon C \to \mathbb{R}$ være en deriverbar funksjon definert i en åpen mengde $C \subseteq \mathbb{R}^n$. Da er:*

(i) f er konveks.

(ii) $f(x) \geq f(x_0) + \nabla f(x_0)^T(x - x_0)$ for alle $x, x_0 \in C$.

(iii) $(\nabla f(x) - \nabla f(x_0))^T(x - x_0) \geq 0$ for alle $x, x_0 \in C$.

Korollar 2.6. *La $f\colon \mathbb{C} \to \mathbb{R}$ være en deriverbar konveks funksjon definert på en åpen konveks mengde $C \subseteq \mathbb{R}^n$. La $x^* \in C$. Da er*

(i) x^* er et lokalt minimum for f.

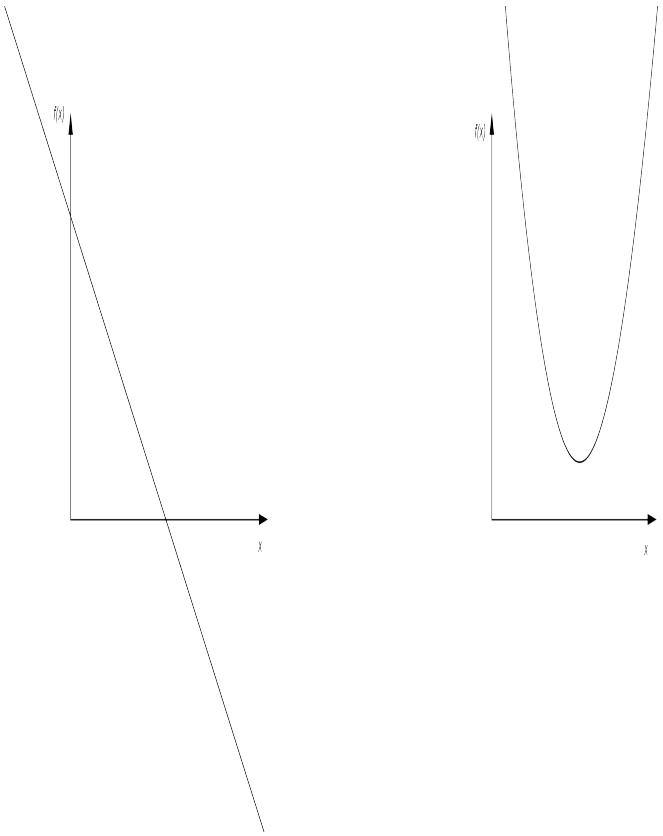
(ii) x^* er et globalt minimum for f.

(iii) $\nabla f(x^*) = 0$.

Gitt et minimeringsproblem, videre skal vi fokusere på betingelsen som karakteriserer løsningen på problemet. Denne betingelsen avhenger av at funksjonen $f\colon \mathbb{C} \to \mathbb{R}$ er konveks og definert på en konveks mengde $C \subseteq \mathbb{R}^n$. Dessuten er funksjonen kontinuerlig deriverbar.
Optimeringsbetingelse for problemet er

Teorem 2.7. *La $x^* \in C$. Da er x^* et lokalt (og dermed et globalt) minimum for f over C hvis og bare hvis*

$$\nabla f(x^*)^T(x - x^*) \geq 0 \quad \textit{for alle} \quad x \in C. \tag{2.10}$$



Figur 4: Noen konvekse funksjoner.

Definisjon 2.8. *Et polyeder $P \subseteq \mathbb{R}^n$ er en løsningsmengde av et system av lineære ulikheter:*

$$P = \{x \in \mathbb{R}^n : Ax \leq b\},$$

hvor A er en reel $m \times n$ matrise, $b \in \mathbb{R}^m$ er en vektor , der ulikheten holder for hver komponent.

La $C \subseteq \mathbb{R}^n$ være en vilkårlig mengde. Vi definerer *konveks hull* av C som en mengde av alle konvekse kombinasjoner av punkter i C.

Definisjon 2.9. *En mengde $P \subset \mathbb{R}^n$ kalles en polytop hvis den er en konveks hull av endelig antall punkter i $\mathbb{R}^n$.*

Lemma 2.10 (Et lokalt minimum for LP er globalt.)**.** *Enhver løsning til et lineært optimeringsproblem som er en lokal minimumsløsning, er en global minimumsløsning.*

Dette lemma innebærer at dersom det finnes et lokalt minimum av et konvekst optimaliseringsproblem, vil dette punktet faktisk være den globale løsningen av problemet. Altså, hvis funksjonen f er konveks, så er det alltid sant (se for mer [Dahl04]) at en lokal optimal løsning er global optimal. En konveks kombinasjon av optimale tillatte løsninger til LP er en konveks polytop.

2.2.1 Løsning av lineære programmer.

Det er ingen enkel analytisk formel for løsning av et lineært program, men det er en rekke meget effektive metoder for å løse dem, inkludert Dantzig's simpleks metoden, og den nyere indrepunktmetoden som beskrives senere i denne oppgaven. Selv om vi ikke kan gi det eksakte antallet av aritmetiske operasjoner som kreves for å løse et lineært program, kan vi bestemme strenge grenser på antall operasjoner som kreves for å løse et lineært program, til en gitt nøyaktighet, ved hjelp av indrepunktsmetoden.

3 Simpleksalgoritme.

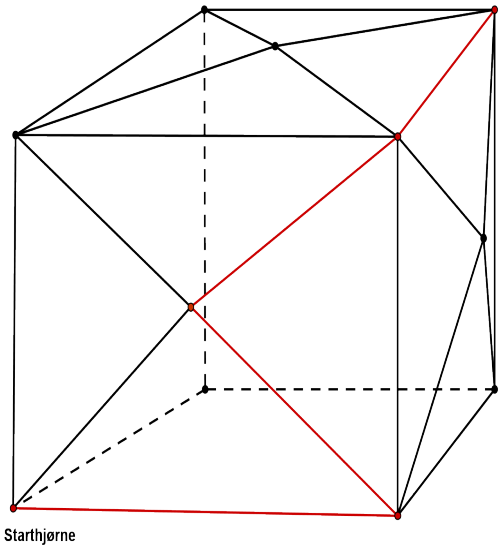
Simpleks metoden er det mest kjente og brukte metoden i praksis for å løse det generelle problemet med lineær programmering (LP).

3.1 Geometri.

Metoden bruker konseptet med *en simpleks* [2], som er *en polytope* av $N+1$ hjørner i N dimensjoner, f.eks. et linjestykke i én dimensjon, en trekant i to dimensjoner og så videre.
Betrakt et LP problem,

$$\begin{aligned} \text{maksimer} \quad & c^T x \\ \text{f.at} \quad & \\ & Ax \leq b \\ & x \geq 0. \end{aligned}$$

hvor $x = (x_1, x_2, \ldots, x_n)$ er variabler, $c = (c_1, c_2, \ldots, c_n)$, A er en $m \times n$ matrise og $b = (b_1, b_2, \ldots, b_m)$ er de lineære begrensningene.
I geometrisk form spesifiserer hver ulikhet et halvrom i det n-dimensjonale euklidske rommet, og krysset er mengden av alle tillatte verdier variablene kan ta.



Figur 5: Viser en polytop sammen med en mulig vei (rød) valgt av simpleks metoden.

Området er konvekst og enten tomt, ubegrenset eller en polytop. Vi lar vår objektivfunksjon få en startverdi v_0 som er definert av hyperplanet $c^T x = v_0$. Vi ser etter den største v slik at hyperplanet fortsatt krysser tillattområdet.

[2] mer om en simpleks i en nettbasert matematisk ordbok ://web.archive.org /web/20070207021813/members.aol.com/Polycell/glossary.html

Om v øker, forflytter hyperplanet seg i retning av vektoren c. Endepunktene på kanten eller en flate vil oppnå den optimale verdien v. Dermed vil den optimale verdien alltid nås i ett av hjørnene i en polytop.
Simpleksalgoritmen begynner på et starthjørne[Karl91] og beveger seg langs kantene av den (muligens ubegrensede) polytop til hjørnene med høyere verdi til objektivfunksjonen. Konveksiteten gir at når et lokalt maksimum er nådd, er det også det globale maksimum, og algoritmen avsluttes. Algoritmen slutter også når en ubegrenset kant er besøkt, og vi kan konkludere med at problemet ikke har noen løsning. Algoritmen vil alltid terminere fordi antallet noder i en polytop er endelig, dessuten siden vi beveger oss alltid i samme retning, håper vi at antall besøkte noder vil være små. Vanligvis er det mer enn ett tilstøtende toppunkt som forbedrer objektivfunksjon, så en pivotregel angir hvilke toppunktet som skal velges. Valget av denne pivotregelen har en stor innvirkning på kjøring av algoritmen, og gir en lang rekke varianter av simpleksmetoden [3].

3.2 Pivotering.

Starter med et eksempel.

Eksempel 1.

$$\begin{aligned} \textit{maksimer} \quad & 3x_1 + 2x_2 \\ \textit{f.at} \quad & \\ & 2x_1 + x_2 \quad \leq 8 \\ & 3x_1 - x_2 \quad \leq 10 \\ & -x_1 + 2x_2 \quad \leq 6 \\ & x_1, x_2 \quad \geq 0 \end{aligned}$$

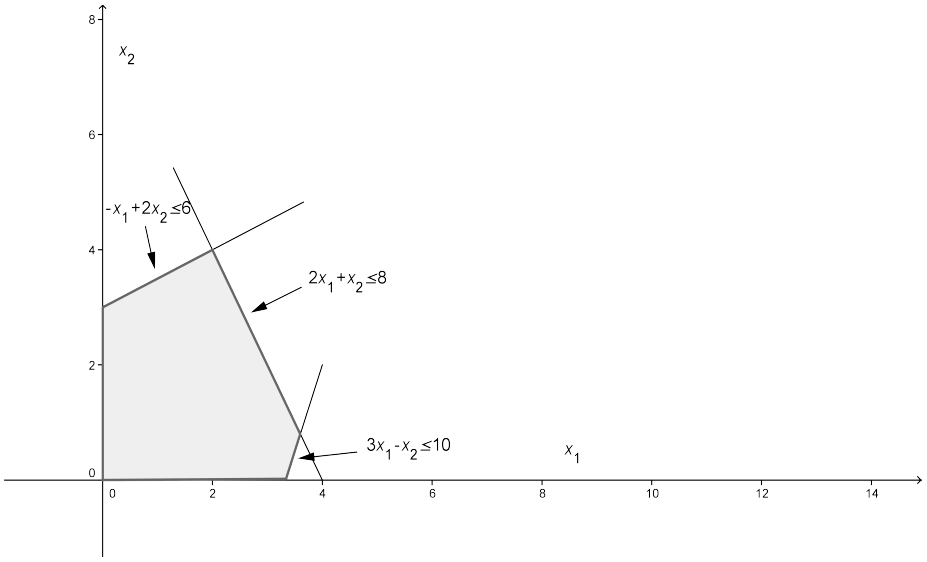
Innfører slakkvariable for hver ulikhet for å konvertere ulikheter til likninger. Da får problemet på *basislisteform:*

$$\begin{aligned} \text{maksimer} \quad & \eta = 3x_1 + 2x_2 \\ \textit{f.at} \quad & \\ & w_1 = 8 - 2x_1 - x_2 \\ & w_2 = 10 - 3x_1 + x_2 \\ & w_3 = 6 + x_1 - 2x_2 \\ & x_1, x_2, w_1, w_2, w_3 \geq 0 \end{aligned}$$

Avhengige variable til venstre kalles *basisvariable*, og uavhengige variable på høyre siden er *ikkebasisvariable*. For å finne en startløsning lar vi alltid ikkebasisvariablene være 0. Startløsningen vi får er ikke optimal siden vi kan øke x-ene. Utfordringen er å finne en maksimal økning av f.eks. x_1 uten at basisvariablene blir negative. Vi får da at $x_1 \leq 4$, $x_1 \leq \frac{10}{3}$:

η	$=$			$3x_1$	$+$	$2x_2$
w_1	$=$	8	$-$	$2x_1$	$-$	x_2
w_2	$=$	10	$-$	$3x_1$	$+$	x_2
w_3	$=$	6	$+$	x_1	$-$	$2x_2$

[3] pivotregel er en regel som sier hvilken inngående variabel som skal velges når det fins valgmuligheter. Det finnes flere pivoteringsregler, bla.: Bland's regel, leksikografiske regelen, bratteste kant regelen og beste forbedringsregelen.



Figur 6: Tillatte basisløsninger er hjørner i en polyeder.

Så vi kan øke x_1 til den minste verdien, nemlig $\frac{10}{3}$ som gir $\eta = 10$ som er mye bedre løsning, men ikke optimal.

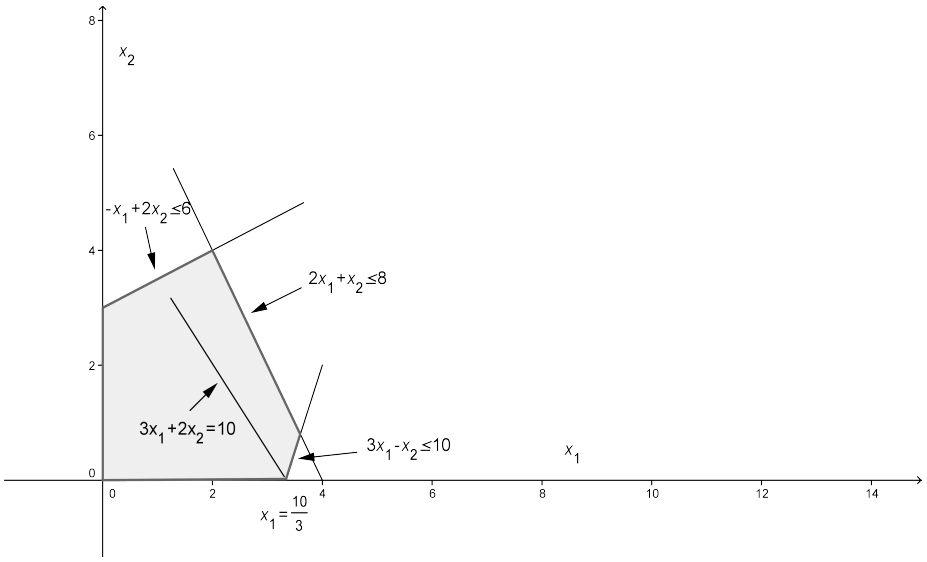
Vi lar x_1 gå inn i basisen og w_2 ut av basisen:

η	$=$	10	-	w_2	$+$	$3x_2$
x_1	$=$	$\frac{10}{3}$	-	$\frac{1}{3}w_2$	$+$	$\frac{1}{3}x_2$
w_1	$=$	$\frac{4}{3}$	$+$	$\frac{2}{3}w_2$	-	$\frac{5}{3}x_2$
w_3	$=$	$\frac{28}{3}$	-	$\frac{1}{3}w_2$	-	$\frac{5}{3}x_2$

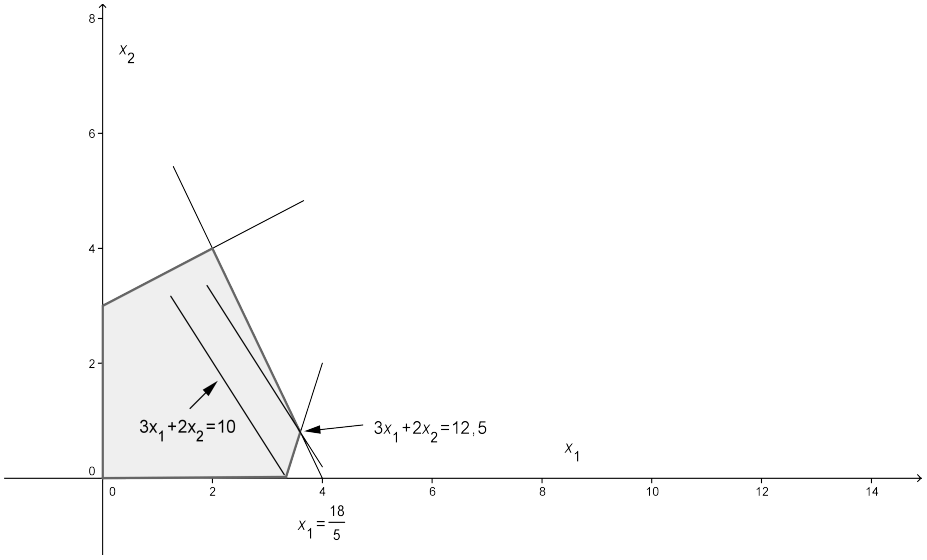
Ny pivotering: x_2 skal inn i basisen og w_1 ut. Dette gir:

η	$=$	$\frac{62}{5}$	-	$\frac{9}{5}w_1$	$+$	$\frac{1}{5}w_2$
x_1	$=$	$\frac{18}{5}$	-	$\frac{1}{5}w_1$	$+$	$\frac{3}{15}w_2$
x_2	$=$	$\frac{4}{5}$	-	$\frac{3}{5}w_1$	$+$	$\frac{2}{5}w_2$
w_3	$=$	8	$+$	w_1	-	w_2

Vi har funnet en ny løsning som er $(x_1,x_2) = (\frac{4}{5},\frac{18}{5})$. Dette gir *en verdi* på η som er $\frac{62}{5}$. Den er fortsatt ikke optimal, så vi beveger oss langs kanten til neste hjørne. Vi kan øke w_2 og få en økning på objektiv funksjonen. Neste pivotering gir:



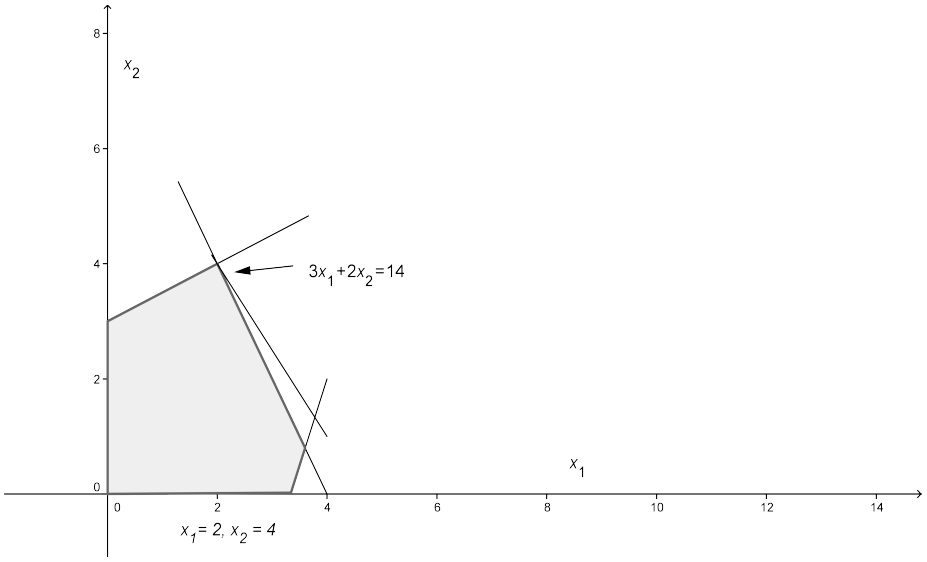
Figur 7: 1.pivotering: vi beveger oss langs kanten $x_2 = 0$. Figuren viser nivåkurven til objektivfunksjonen η som stadig forskyves inntil den tangerer polyederen.



Figur 8: 2.pivotering: løsningen er fortsatt ikke optimal.

$$\begin{array}{lrrrrrr} \eta & = & 14 & - & \frac{8}{5}w_1 & - & \frac{1}{5}w_3 \\ x_1 & = & 2 & - & \frac{2}{5}w_1 & + & \frac{3}{15}w_3 \\ x_2 & = & 4 & - & \frac{1}{5}w_1 & + & \frac{2}{5}w_3 \\ w_2 & = & 8 & + & w_1 & - & w_3 \end{array}$$

Endelig har alle variablene i objektivfunksjonen et negativt fortegn, dette gir at løsningen vi har funnet er optimal med $\eta = 14$ (figur (9)).



Figur 9: Løsningen er optimal.

3.3 Numeriske utfordringer.

3.3.1 To faser problemer.

Gitt LP problem:

$$\begin{array}{rll} \text{maksimer} & \sum_{j=1}^{n} c_j x_j & \\ \text{f.at} & & \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for} \quad i = 1, 2, \ldots, m \\ & x_j \geq 0 & \text{for} \quad j = 1, 2, \ldots, n. \end{array}$$

Vi innfører slakkvariable og får problemet på basislisteform:

$$\begin{array}{rcl} \eta & = & \sum_{j=1}^{n} c_j x_j \\ w_i & = & b_i - \sum_{j=1}^{n} a_{ij} x_j \quad \text{for} \quad i = 1, 2, \ldots, m. \end{array}$$

Vi finner en initiell tillatt løsning ved å la hver x_j være lik 0 og sette hver w_i lik korresponderende b_i. Denne løsningen er tillatt hvis og bare hvis alle $b_i \geq 0$. Men hva hvis noen b_i-er er negative? Vi løser dette problemet ved å introdusere et nytt *hjelpeproblem* som er slik at

1. tillatt basisliste er lett å finne
2. optimal basisliste gir tillattliste for originalproblem.

Det nye hjelpeproblemet kalles *Fase I* problem:

$$\begin{array}{rll} \text{maksimer} & -x_0 & \\ \text{f.at} & & \\ & \sum_{j=1}^{n} a_{ij} x_j - x_0 \leq b_i & \text{for} \quad i = 1, 2, \ldots, m \\ & x_j \geq 0 & \text{for} \quad j = 1, 2, \ldots, n. \end{array}$$

Det er lett å finne en tillattløsning for hjelpeproblemet. I Fase I innfører vi slakkvariable på vanlig måte, og konverterer ikke-tillattbasisliste til en tillattbasisliste ved at vi lar den mest negative variabelen forlate basisen. Da basislisten til hjelpeproblemet er optimal, dropper vi x_0 fra likningene og rekonstruerer original objektiv funksjon(se for mer referanse [Vand08]). Denne siste prosessen fra tillattløsning til optimal løsning kalles *Fase II.*

3.3.2 Ubegrenset løsning.

Hvordan vet vi når objektivfunksjonen er ubegrenset? Vi skal se nærmere på pivotering: velg l fra $\{i \in \mathcal{B} : \bar{a}_{ik}/\bar{b}_i \, er \, maksimal\}$. Det kan tenkes at alle $a_{ik}/\bar{b}_i$ er ikkepositive. I dette tilfellet vil ingen av basisvariablene bli null. Inngående variable kan økes vilkårlig mye og produsere vilkårlig stor verdi på objektivfunksjonen. I slike situasjoner sier vi at problemet er *ubegrenset.*

3.3.3 Degenerasjon og sirkling.

Vi kaller basislisten *degenerert* dersom $\bar{b}_i = 0$ for minst en i. Vi har da en *degenerert* basisløsning. Hvis en degenerert basisliste gir en degenerert pivotering, kan vi få problemer. Det er ikke nødvendigvis slik at degenererte pivoteringer kan gi problemer, men hvis vi får en sekvens av bare degenererte pivoteringer, så kan vi komme tilbake til samme basis. For da vil denne sekvensen gjentas i det uendelige, og algoritmen vil ikke finne optimal løsning. Dette fenomenet kalles *sirkling.* Et viktig teorem fra Vanderbei [Vand08]:

Teorem 3.1. *Hvis simpleksmetoden ikke terminerer, så må den sirkle.*

I 1977 publiserte professor Robert Bland (Cornell University Engineering) en ny og enkel til å implementere pivoteringsregel for simpleksmetoden. Blands regel sier[Dant97]:

1. Inngående kolonne. Velg pivotkolonne $j = s$ med relativ cost $\bar{c}_j < 0$ som har lavest indeks j.
2. Utgående kolonne. Velg utgående basiskolonne j_r blandt dem som er kvalifisert for dropping med lavest indeks j_i.

Teorem 3.2 (Terminering ved bruk av Bland's regel.)**.** *Sykling er umulig ved bruk av Bland's regel.*

Liten kommentar fra Dantzig: Bland's regel er veldig enkel til å implementere og garantert hindrer sykling. Den største ulempen er at valg av inngående kolonne kan være dårlig. I følge Dahl kan man få en liten økning på objektivfunksjonen i hver pivotering, og dermed mange pivoteringer og lang regnetid.
Nå har vi Fase I algoritme og en variant av simpleksmetoden som garantert terminerer, som gjør at vi kan oppsummere med et følgende teorem:

Teorem 3.3 (Fundamentalteoremet i LP.)**.** *For hvert vilkårlig LP problem gjelder:*

- *Hvis det ikke finnes en optimal løsning, så er problemet enten ikketillatt eller ubegrenset.*
- *Hvis tillatt løsning eksisterer, så fins en tillatt basisløsning.*
- *Hvis optimal løsning eksisterer, så fins en optimal basisløsning.*

3.4 Dualitet.

Gitt LP problem (P) i standardform:

$$\begin{array}{rll} \text{maksimer} & \sum_{j=1}^{n} c_j x_j & \\ \text{f.at} & & \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for} \quad i = 1, 2, \ldots, m \\ & x_j \geq 0 & \text{for} \quad j = 1, 2, \ldots, n. \end{array}$$

Vi kaller (P) *primal* problem. Til ethvert primal problem er det knyttet et annet omvendt LP problem som kalles det *dualet* (D) problemet til (P) som gir en øvre grense for den optimale verdien av primal problemet. I matriseform kan vi uttrykke det dualet problemet slik::

$$\begin{array}{rll} \text{minimer} & \sum_{i=1}^{m} b_i y_i & \\ \text{f.at} & & \\ & \sum_{i=1}^{m} y_i a_{ij} \geq c_j & j = 1, 2, \ldots, n \\ & y_i \geq 0 & i = 1, 2, \ldots, m. \end{array}$$

Von Neumann [4] formulerte uten å bevise dualitets teorem: *Hvis primal problem og dual problem har tillatte løsninger, så eksisterer det en optimal tillattløsning til dem begge to som er like.*

Teorem 3.4 (Infeasibility teorem.)**.** *Et sett av lineære ulikheter [Dant97]*

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_j \quad \textit{for} \quad i = 1, 2 \ldots, m$$

er ikketillatt hvis og bare hvis det fins en ikkenegativ kombinasjon av ulikheter som er en ikketillatt ulikhet. I matrisenotasjon er systemet $Ax \leq b$ ikketillatt hvis og bare hvis det fins en vektor $y \geq 0$ slik at $y^T Ax \leq y^T b$ er en ikketillatt ulikhet, nemlig $y^T A = 0$ og $y^T b > 0$.

Korollar 3.5 (Ikketillatt likhet.)**.** *Hvis et lineært likningssystem med positive variable er ikketillatt, så fins det en lineær kombinasjon av likninger som er en ikketillatt likning med positive variable.*

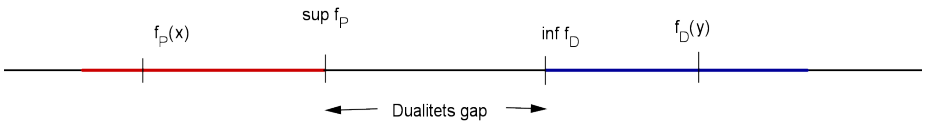
Anta at det fins primal og dual løsning, svak form av dualitets teorem er følgende:

Teorem 3.6 (Svak dualitet.)**.** *Hvis $(x_1, x_2, \ldots, x_n)$ er tillatt i (P) og $(y_1, y_2, \ldots, y_m)$ er tillatt i (D) har vi*

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i$$

Betrakt delmengde av reelle linjen bestående av alle mulige verdier for primal objektivfunksjon, og betrakt tilsvarende delmengde knyttet til dualproblem. Svak dualitetsteoremet forteller oss at mengden av primalverdier ligger helt til venstre for mengde av duale verdier. Disse mengdene er begge lukkede intervaller, og høyre endepunkt for primal mengde er venstre endepunkt for dual mengde. Det vil si at det ikke er gapet mellom optimal objektivfunksjonsverdi for primal og dual problem (se figur 10).

[4] fundamental begrep dualitet og terminologi ble introdusert av John von Neumann i samtale med George Danzig i oktober 1947 og vises implisitt i et notat han skrev noen uker senere (les mer i [Dant97])



Figur 10: Illustrasjon av dualitets gap. Primal objektivfunksjonsverdi er mindre enn dual objektivverdi. Spørsmålet er om det fins gap mellom største primalverdi og minste dualverdi.

Mangelen på et gap mellom primal og dual objektivverdier gir et praktisk verktøy for å verifisere optimalitet. Hvis vi kan vise at primal tillattløsning $(x_1^*, x_2^*, \ldots, x_n^*)$ og dual tillattløsning $(y_1^*, y_2^*, \ldots, y_m^*)$ for hver

$$\sum_{j=1}^{n} c_j x_j^* = \sum i = 1^m b_i y_i^*,$$

kan vi konkludere med at hver av disse løsningene er optimal for det respektive problemet. Det faktumet at det aldri er noe gap mellom primal og dual optimal objektivverdi er vanligvis referert til *Sterk Dualitets teorem*[Vand08]:

Teorem 3.7 (Sterk dualitet.)**.** *Hvis primal problem har en optimal løsning* $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$*, så har dual en optimal løsning* $y^* = (y_1^*, y_2^*, \ldots, y_m^*)$ *slik at*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

Konsekvensen av sterk dualitet er at (P) og (D) har samme optimal verdi når (P) har optimal løsning. Samtidig er det nødvendig å gjenopprette en optimal dual løsning når bare en optimal primal løsning er kjent. Følgende teorem kjent som *Komplementær slakk teorem* tar hensyn til dette [Vand08]:

Teorem 3.8 (Komplementær slakk.)**.** *Anta at* $x = (x_1, x_2, \ldots, x_n)$ *er primal tillatt og* $y = (y_1, y_2, \ldots, y_m)$ *er dual tillatt. La* $(w_1, w_2, \ldots, w_m)$ *være tilhørende primale slakkvariable, og* $(z_1, z_2, \ldots, z_n)$ *tilhørende duale slakkvariable. Da er* x *og* y *optimale for respektive problemer hvis*

$$\begin{aligned} x_j z_j &= 0, \quad \textit{for} \quad j = 1, 2, \ldots, n, \\ w_i y_i &= 0, \quad \textit{for} \quad i = 1, 2, \ldots, m. \end{aligned}$$

I følge Danzig og Thara (se for referanse [Dant97], hvis det er en slakk i en ulikhet (som betyr at slakkvariabelen, f.eks. $w_i > 0$) i et av problemene, så må den tilhørende dualvariabelen være null.

3.5 Primal simpleksalgoritme i matriseform.

3.5.1 Notasjon.

Betrakt som vanlig et generelt LP problem:

$$\begin{array}{rll} \text{maksimer} & \sum_{j=1}^{n} c_j x_j & \\ \text{f.at} & & \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for} \quad i = 1, 2, \ldots, m \\ & x_j \geq 0 & \text{for} \quad j = 1, 2, \ldots, n. \end{array}$$

Vi skiller ikke slakkvariable og originale variable, så vi introduserer slakkvariable på følgende måte:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1, 2, \ldots, m.$$

Vi kan nå konvertere ulikheter til likninger, og LP problemet på matriseform ser slik ut:

$$\begin{array}{rl} \text{maksimer} & c^T x \\ \text{f.at} & \\ & Ax = b \\ & x \geq 0. \end{array}$$

La $\mathcal{B}$ være indeksmengden til basisvariablene og $\mathcal{N}$ være indeksmengden til ikkebasisvariablene. Nå kan hver i-te komponent i Ax deles opp i basisdel og ikkebasisdel:

$$\sum_{j=i}^{n+m} a_{ij} x_j = \sum_{j \in \mathcal{B}} a_{ij} x_j + \sum_{j \in \mathcal{N}} a_{ij} x_j.$$

Innfører følgende notasjon for disse to delene [Vand08]: lar B være en $m \times m$ submatrise som har eksakt m kolonner fra A assosiert med basisvariable, og tilsvarende N være en $m \times n$ submatrise som har n ikkebasiskolonner fra A. Matrise A i partisjonert matriseform ser slik ut:

$$A = \begin{bmatrix} B & N \end{bmatrix}$$

Tilsvarende splitter x:

$$x = \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix}.$$

Med den nye notasjonen blir

$$Ax = \begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix} = Bx_{\mathcal{B}} + Nx_{\mathcal{N}}.$$

Tilsvarende partisjonering av c gir:

$$c^T x = \begin{bmatrix} c_{\mathcal{B}} \\ c_{\mathcal{N}} \end{bmatrix}^T \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix} = c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}}.$$

3.5.2 Matematisk beskrivelse.

Likningen $Ax = b$ kan skrives som

$$Bx_{\mathcal{B}} + Nx_{\mathcal{N}} = b.$$

Siden matrise B er invertibel[5], kan vi skrive $x_{\mathcal{B}}$ som en funksjon av ikkebasisvariable $x_{\mathcal{N}}$:

$$x_{\mathcal{B}} = B^{-1}b - B^{-1}Nx_{\mathcal{N}}. \tag{3.1}$$

Tilsvarende kan objektivfunksjonen ζ skrives som

$$\begin{aligned} \zeta &= c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}} && (3.2)\\ &= c_{\mathcal{B}}^T(B^{-1}b - B^{-1}Nx_{\mathcal{N}}) + c_{\mathcal{N}}^T x_{\mathcal{N}} && (3.3)\\ &= c_{\mathcal{B}}^T B^{-1}b - \left((B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}\right)^T x_{\mathcal{N}}. \end{aligned}$$

Nå kan basislisteformen vi har brukt hittil omskrives slik:

$$\begin{array}{rcrcr} \eta & = & c_{\mathcal{B}}^T B^{-1}b & - & \left((B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}\right)^T x_{\mathcal{N}} \\ \hline x_{\mathcal{B}} & = & B^{-1}b & - & B^{-1}Nx_{\mathcal{N}}, \end{array}$$

hvor

$$\begin{aligned} c_{\mathcal{B}}^T B^{-1}b &= \bar{\zeta} \\ c_{\mathcal{N}} - (B^{-1}N)^T c_{\mathcal{B}} &= [\bar{c_j}] \\ B^{-1}b &= [\bar{b_i}] \\ B^{-1}N &= [\bar{a_{ij}}], \end{aligned}$$

der har vi på høyre side indeksert vektorer og matriser med $i \in B$ og $j \in N$. Basisløsningen knyttet til basislisten er

$$\begin{aligned} x_{\mathcal{N}}^* &= 0, \\ x_{\mathcal{B}}^* &= B^{-1}b. \end{aligned} \tag{3.4}$$

Dual basislisten knyttet til primal basislisten er i simpelhet den negativ-transponerte av primal basislisten. Vi innfører duale slakkvariable på samme måte som primale, og minner om en korrespondanse mellom duale og primale variable:

- primal variabel x_j svarer til dual slakkvariabel z_j,
- primal slakkvariabel w_i svarer til dual variabel y_i.

Vi sier at x_j og z_j er *komplementære*, og at w_i og y_i er *komplementære*. Tilsvarende utregning for duale variable gir dual basislisteform:

$$\begin{array}{rcrcr} -\xi & = & -c_{\mathcal{B}}^T B^{-1}b & - & (B^{-1}b)^T z_{\mathcal{B}} \\ \hline z_{\mathcal{N}} & = & (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}} & + & (B^{-1}N)^T z_{\mathcal{B}}. \end{array}$$

[5]med B er invertibel menes at kolonnene i B er m lineært uavhengige vektorer i $\mathbb{R}^m$ som kalles en basis

Den duale basisløsningen knyttet til basislisten over er

$$\begin{aligned} z^*_{\mathcal{B}} &= 0, \\ z^*_{\mathcal{N}} &= (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}. \end{aligned} \tag{3.5}$$

Bruker (3.4) og (3.5) og innfører nå

$$\zeta^* = c_{\mathcal{B}}^T B^{-1} b, \tag{3.6}$$

slik at vi kan skrive primal og dual basisliste på formen:

$$\begin{array}{rcrcl} \zeta & = & \zeta^* & - & (z^*_{\mathcal{N}})^T x_{\mathcal{N}} \\ \hline x_{\mathcal{B}} & = & x^*_{\mathcal{B}} & - & B^{-1}N x_{\mathcal{N}} \\ \hline -\xi & = & -\zeta^* & - & (x^*_{\mathcal{B}})^T z_{\mathcal{B}} \\ \hline z_{\mathcal{N}} & = & z^*_{\mathcal{N}} & + & (B^{-1}N)^T z_{\mathcal{B}}. \end{array}$$

3.5.3 En iterasjon i simpleksalgoritmen.

Et step i simpleks metoden kalles *en iterasjon* [Vand08].

Step 1. *Test optimalitet.* Stopp hvis $z^*_{\mathcal{N}} \geq 0$. Da er nåværende basisløsningen optimal.

Step 2. *Velg inngående variabel.* Velg en indeks $j \in \mathcal{N}$ for $z^*_j < 0$. Variabel x_j er *inngående variabel.*

Step 3. *Beregn primal skrittretning* $\Delta x_{\mathcal{B}}$. Vi har

$$x_{\mathcal{N}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ t \\ 0 \\ \vdots \\ 0 \end{bmatrix} = te_j,$$

der e_j er j-te enhetsvektor som angir endringen i primale ikkebasisvariabler. Da er

$$x_{\mathcal{B}} = x^*_{\mathcal{B}} - B^{-1}Nte_j. \tag{3.7}$$

Skrittretningen er gitt ved

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j. \tag{3.8}$$

Step 4. *Beregn primal skrittlengde.* Vi velger størst mulig $t \geq 0$ slik at $x_{\mathcal{B}}$ er negativ. Vi velger dermed størst t for

$$x^*_{\mathcal{B}} \geq t\Delta x_{\mathcal{B}}.$$

For hver $i \in \mathcal{B}^*$, $x_i^* \geq 0$ og $t \geq 0$ kan vi dividere begge sider av de overnevnte ulikhetene ved disse tallene og bevare ulikheten. Vi får at

$$\frac{1}{t} \geq \frac{\Delta x_i}{x_i^*}, \qquad \text{for} \quad \forall i \in \mathcal{B}$$

Vi vil la t være så stor som mulig, og $1/t$ så liten som mulig. Den minste mulige verdi for $1/t$ som tilfredsstiller alle de nødvendige ulikheter er åpenbart

$$\frac{1}{t} = \max_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*}.$$

Dermed er den maksimale t gitt ved

$$t = \left(\max_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*} \right)^{-1}. \tag{3.9}$$

Hvis den maksimale verdien er mindre eller lik 0, kan vi stoppe her og primale er ubegrenset.

Step 5. *Velg utgående basisvariabel.* Velg en indeks i der maksimumet inntraff i (3.9), og la x_i være utgående basisvariabel.

Step 6. *Beregn dual skrittretning* $\Delta z_{\mathcal{N}}$. Dette finner vi når vi ser på dual basisliste:

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i.$$

Step 7. *Beregn dual skrittlengde.* Siden z_j forlater basis får vi skrittlengde s:

$$s = \frac{z_j^*}{\Delta z_j}.$$

Step 8. *Oppdater primal og dual løsning.* Følgende oppdateringer skal utføres:

$$\begin{aligned} x_j^* &\leftarrow t \\ x_{\mathcal{B}}^* &\leftarrow x_{\mathcal{B}}^* - t\Delta x_{\mathcal{B}} \end{aligned}$$

$$\begin{aligned} z_i^* &\leftarrow s \\ z_{\mathcal{N}}^* &\leftarrow z_{\mathcal{N}}^* - s\Delta z_{\mathcal{N}} \end{aligned}$$

.

Step 9. *Oppdaterer basis.* Tilslutt oppdateres basis:

$$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}.$$

3.6 Effektivitet.

Simpleksalgoritmen har en ***while loop***, og det er vanskelig å gi noen anslag over en øvre grense for antall iterasjoner av en while loop. Vi kan si noe om antallet operasjoner i en iterasjon. I forrige seksjon kom vi fram til følgende to viktige resultater:

Algorithm 1 Primal simpleksalgoritme.

Anta $x^*_{\mathcal{B}} \geq 0$
while $(z^*_{\mathcal{N}} \ngeq 0)$ **do**
  pick $j \in \{j \in \mathcal{N}\colon z^*_j < 0\}$
  $\Delta x_{\mathcal{B}} = B^{-1}Ne_j$
  $t = \left(\max_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i}\right)^{-1}$
  pick $i \in argmax_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i}$
  $\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$
  $s = \frac{z^*_j}{\Delta z_j}$
  $x^*_j \leftarrow t$
  $x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t\Delta x_{\mathcal{B}}$
  $z^*_i \leftarrow s$
  $z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s\Delta z_{\mathcal{N}}$
  $\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$
end while

- Hvis en LP har en avgrenset optimal løsning, da eksisterer et ekstremt punkt i tillattområdet som er optimal.
- Ekstreme punkter i tillattområdet til LP tilsvarer tillatte basisløsninger av LP.

Den første av disse resultatene sier at for å oppnå en optimal løsning av en LP, kan vi begrense våre søk til mengden av ekstreme punkter i tillattområdet. Det andre resultatet gir en algebraisk karakterisering av denne mengden: hvert av disse punktene bestemmes ved å velge et sett av basisvariable med kardinalitet lik antall av de begrensningene på LP, og det ekstra kravet er at verdier av disse variablene er ikkenegative.
Den siste observasjonen ville få en til å tenke at en tilnærming til problemet ville være å nummerere hele mengden av ekstreme punkter, sammenligne tilsvarende objektivverdier, og til slutt velge en som maksimerer/minimerer objektivfunksjon over denne mengden.
En slik tilnærming ville egentlig fungere for ganske små problemer. Men for rimelig store LP, kan mengde av ekstreme punkter bli svært stor. Det første tilfelle av en ikke-triviell LP løst med simpleksalgoritmen ble muligens Ladermans løsning. Dette LP hadde 9 begrensninger og 77 variabler. Angivelig jobbet ni medarbeidere med elektroniske kalkulatorer totalt 120 arbeidsdager for å utføre disse beregningene. Den første gjennomføringen av simpleks metoden med datamaskin har vært utviklet ved National Bureau of Standards, i dag National Institute of Standards and Technology, på SEAC datamaskinen. LP med 48 ligninger og 71 variabler ble løst på 18 timer og omfattet 73 simpleks iterasjoner.
Hvordan kan vi måle hvor rask LP av en gitt størrelse kan løses? Det er to typer mål på effektiviteten av en algoritme:

- verste tilfelle: som navnet sier ser vi på alle problemer av en gitt størrelse, og spør hvor mye innsats er nødvendig for å løse den hardeste av disse

problemene,

- gjennomsnitt: ser på gjennomsnittlig arbeidsmengde, gjennomsnitt over alle problemer av en gitt størrelse.

Verste tilfelle analyse er generelt lettere enn gjennomsnitt analyse. I verste tilfelle analyse trenger man bare å oppgi en øvre grense på hvor mye innsats kreves, og deretter vise et konkret eksempel som oppnår denne grensen. Men for gjennomsnitt analyse må man ha tilfeldige LP problemer og da kan man si noe om gjennomsnitt eller forventet antall over regneoperasjoner.

Siden simpleksmetoden (uten sykling) fungerer ved å flytte fra én tillatt basisløsning til en annen uten å vende tilbake til tidligere besøkte løsninger, kan en øvre grense på antall gjentakelser være ganske enkelt. Antall tillatte basisløsninger kan være:

$$\binom{n+m}{m}.$$

Det "verste eksempelet" er i dette tilfelle når $m = n$. Da er

$$\frac{1}{2n} \leq \binom{2n}{n} \leq 2^{2n}.$$

I 1972 fant V.Klee og G.J.Minty en klasse av LP hvor simpleksmetoden bruker største koeffisientregel på formen:

$$\begin{array}{rll} \text{maksimer} & \sum_{j=1}^{n} 10^{n-j} x_j & \\ \text{f.at} & & \\ & 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1} & \text{for} \quad i = 1, 2, \ldots, m \\ & x_j \geq 0 & \text{for} \quad j = 1, 2, \ldots, n. \end{array}$$

Det viser seg at antall pivoteringer blir $2^n - 1$.
Enorme økninger i datamaskinens minnekapasitet i de siste 10-årene har gjort det mulig å håndtere mye større problemer, og har også gjort det mulig å vurdere helt annerledes løsningsstrategier og implementeringer av disse strategiene. Mange av de grunnleggende algoritmiske ideer som er nøkkelen til moderne LP koder, kunne ikke ha blitt gjennomført hvis det ikke var nok minne. Forbedringene i dataprogrammeringspråk og systemer har også gjort det mye enklere å bygge store og komplekse systemer. Forbedringer i datamaskin-menneske-grensesnitt har bedret ikke bare brukbarheten av verktøyene, men også verktøyene selv, inkludert grunnleggende forbedringer i de underliggende algoritmer. Situasjonen for barriere algoritmer er noe annerledes enn for simpleksalgoritmer. I barriere algoritmer er det et enkelt beregningsorientert skritt som vanligvis dominerer: beregning av Cholesky factorisasjon. Det finnes ingen slike enkelte trinn som dominerer i simpleksberegninger. I tillegg til at bruk av Cholesky factoriseringen er veldig vanlig, så utnyttes det mulighetene som moderne databehandlings verktøyene har, bedre enn vi gjør det i simpleksalgoritmer(se for mer referanse [Bixb]).

4 Indrepunktsmetode for LP.

I denne oppgaven studeres et alternativ til simpleksmetoden for løsning av lineære programmerings problemer. Denne algoritmen kalles *path-following method* (her kalt *PF-metoden*). Denne hører til en klasse som heter *indrepunktsmetoder*. Indrepunktmetoder (også referert til som *barriere metoder*) er en bestemt gruppe algoritmer til å løse lineære og ikke-lineære konvekse optimeringsproblemer. Disse algoritmene har blitt inspirert av Karmarkar algoritme, utviklet av Narendra Karmarkar i 1984, for lineær programmering.
Det teoretiske grunnlaget for indrepunktsmetoder består av tre avgjørende byggesteiner ([Mars90]. Først, Newton's (1687) metode for løsning av ikkelineære likninger. Andre, Lagrange's (1788) metode for optimering med lineære begrensninger. Tredje, Fiacco og McCormick's (1968) barriere metode for optimering med begrensninger som er ulikheter.

4.1 Barriere problem.

Betrakt LP problem som er uttrykt som vanlig, ved hjelp av ulikheter og ikkenegative variabler:

$$\begin{aligned} \text{maksimer} \quad & c^T x \\ \text{f.at} \quad & \\ & Ax \leq b \\ & x \geq 0. \end{aligned}$$

Tilsvarende dual problem er

$$\begin{aligned} \text{minimer} \quad & b^T y \\ \text{f.at} \quad & \\ & A^T y \geq c \\ & y \geq 0. \end{aligned}$$

Vi legger til slakkvariablene til begge problemene som vanlig:

$$\begin{aligned} \text{maksimer} \quad & c^T x \\ \text{f.at} \quad & \\ & Ax + w = b \\ & x, w \geq 0. \end{aligned}$$

og

$$\begin{aligned} \text{minimer} \quad & b^T y \\ \text{f.at} \quad & \\ & A^T y - z = c \\ & y, z \geq 0. \end{aligned}$$

Gitt et begrenset maksimeringsproblem hvor noen av begrensningene er ulikheter, kan en betrakte tilsvarende ulikheter med en ekstra betingelse i en objektiv funksjon. F. eks., vi kan erstatte en begrensning med en variabel x_j, som er ikkenegativ, ved å addere objektiv funksjon med en betingelse som er negativ uendelig når $x_j < 0$ og lik null ellers. Ulempen med den nye funksjonen er at den er diskontinuerlig, noe som hindrer oss å bruke Kalkulus til å studere den. Anta nå at vi erstatter denne diskontinuerlige funksjonen med en annen funksjon som er negativ uendelig når x_j er negativ, men endelig når x_j er positiv, og nærmer seg negativ uendelig når x_j nærmer seg null. Nå kan funksjonen oppfattes som

glatt når vi ser bort fra diskontinuitet, og kan bruke Kalkulus nå. Den enkleste funksjonen som oppfyller disse kravene er en logaritmisk funksjon slik at den nye objektivfunksjonen er:

$$\begin{array}{rl} \text{maksimer} & c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i \\ \text{f.at} & \\ & Ax + w = b \end{array}$$

Det nye problemet kalles *barriere problem*. Merk, det er en hel familie av problemer indeksert ved parameteren μ. Hvert av disse problemene er ikkelineære fordi objektivfunksjonen er ikkelineær. Denne ikkelineære objektivfunksjonen kalles *barriere funksjon* eller i dette tilfellet, *logaritmisk barriere funksjon.*

4.1.1 Geometrisk tolkning.

Vi vet at en mengde av tillatte løsninger for LP problem er polyeder hvor hver flate har egenskapen at en av variablene er null. Derfor er barrierefunksjon en minus uendelig i hver av flatene til en polyeder. Den er endelig i indre delen til polyederen og nærmer seg minus uendelig når vi nærmer oss grensen. For hver μ oppnås maksimum i et indre punkt, og når μ nærmer seg null, flyttes dette indre punktet nærmere optimal løsning for det originale LP problemet. Et sett av optimale løsninger til barriereproblemer danner vei (*path*) gjennom indre punkter av polyeder, som er tillatte løsninger. Denne veien kalles *central path.*

4.2 Lagrangemultiplikator.

Betrakt et generelt optimeringsproblem hvor vi maksimerer en funksjon med en eller flere bibetingelser. Funksjonen kan være ikkelineær, men skal være glatt og to ganger deriverbar. Anta i første omgang at vi maksimerer funksjon med kun en bibetingelse:

$$\begin{array}{rl} \text{maksimer} & f(x) \\ \text{f.at} & g(x) = 0 \end{array}$$

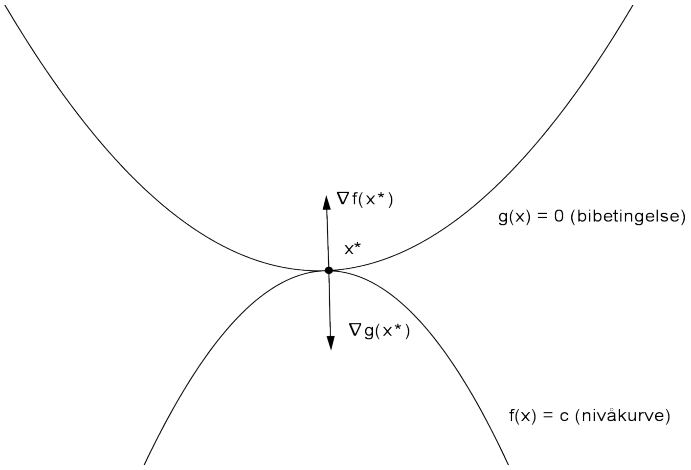
Når vi leter etter våre maksimums- og minimumspunkter, må vi lete etter punkter der nivåkurven [6] tangerer bibetingelseskurven, med andre ord, der normalen til nivåkurven er parallell med normalen til bibetingelseskurven. Gradienten [7] må stå ortogonalt på mengden av tillatte løsninger $\{x : g(x) = 0\}$. Kravet for at et punkt x^* er et ekstremal punkt, er at det er tillatt og at $\nabla f(x^*)$ er proporsjonal med $\nabla g(x^*)$. Altså, det skal finnes et tall y slik at likningssystemet er oppfylt:

$$\begin{array}{rl} \nabla f(x^*) & = y \nabla g(x^*) \\ g(x^*) & = 0. \end{array}$$

Proporsjonalitetskonstanten y kan være et reelt tall, negativt, positivt eller null. Denne proporsjonalitetskonstanten kalles *Lagrangemultiplikator.*
Vi skal nå se på et problem når vi ønsker å optimere en funksjon under flere bibetingelser:

[6] En *nivåkurve* $N_c = \{(x, y) | f(x, y) = c\}$ er en kurve bestående av alle punkter med en gitt funksjonsverdi.

[7] Vi tenker på gradienten som en vektor som peker i den retningen hvor funksjonen vokser raskest og lengden til gradienten er lik stigningstallet i denne retningen.



Figur 11: Punkt a er optimal siden gradient til f er perpendikulær til tillattmengden.

$$\begin{aligned} \text{maksimer} \quad & f(x) \\ \text{f.at} \quad & \\ & g_1(x) = 0 \\ & g_2(x) = 0 \\ & \vdots \\ & g_m(x) = 0 \end{aligned}$$

De likningene $g_1(x) = 0, \ldots, g_m(x) = 0$ vil normalt definere flater i rommet som skjærer hverandre langs en kurve. Problemet er å finne den største verdien til f langs denne kurven. Funksjonen langs kurven stiger i den retningen hvor ∇f peker. Skal vi finne optimalverdien, må ∇f i dette punktet ligge i normalplanet til kurven. Altså, nå ligger punktet x i planet som er utspent av gradienter til flatene. Derfor må ∇f være en lineærkombinasjon av normalvektorene $\nabla g_1(x) = 0, \ldots, \nabla g_m(x) = 0$ til flatene $g_1(x) = 0, \ldots, g_m(x) = 0$. Dette gir følgende likningsystemet:

$$\begin{aligned} \nabla f(x^*) &= \sum_{i=1}^{m} y \nabla g(x^*) \\ g(x^*) &= 0. \end{aligned}$$

Definerer *Lagrangefunksjonen* [Vand08]:

$$L(x, y) = f(x) - \sum_i y_i g_i(x)$$

Regner ut de partiellderiverte til L og setter disse uttrykkene lik null:

$$\begin{aligned} \tfrac{\partial L}{\partial x_j} = \tfrac{\partial f}{\partial x_j} - \sum_i y_i \frac{\partial g_i}{\partial x_j} &= 0, \quad j = 1, 2, \ldots, n, \\ \tfrac{\partial L}{\partial y_i} = -g_i &= 0, \quad i = 1, 2, \ldots, m. \end{aligned}$$

Disse likningene refererer til *førsteordens optimalitetskriterier*. Å bestemme en løsning som virkelig er en optimal løsning ved disse kriterier, kan være vanskelig. Hvis alle begrensningene er lineære, så er det første steget (som ofte er tilstrekkelig) å se på en Hessian-matrise av f i x

$$Hf(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right].$$

Vi har da følgende teorem:

Teorem 4.1. *Hvis begrensningene er lineære, så er punktet x^* et lokalt maksimum hvis*

$$\xi^T Hf(x^*)\xi < 0 \tag{4.1}$$

for hver $\xi \neq 0$ tilfredstiller

$$\xi^T \nabla g_i(x^*) = 0, \quad i = 1, 2, \ldots, m.$$

4.3 Lagrangemultiplikator og barriereproblem.

Vi skal bruke Lagrangemultiplikatorer til å studere løsninger på barriereproblemet, og se at for hver verdi av barriereparameteren μ er det en unik løsning til barriereproblem. Vi vil se at når μ går mot null, vil løsningen til barriereproblemet nærmer seg løsningen på det originale LP problemet. Betrakt et barriereproblem:

$$\begin{array}{rl} \text{maksimer} & c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i \\ \text{f.at} & \\ & Ax + w = b. \end{array}$$

Begrensningene er likninger, så vi kan bruke Lagrangemultiplikatorteknikken for å løse dette problemet. Har Lagrangefunksjonen som ser ut slik:

$$L(x, w, y) = c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i + y^T(b - Ax - w).$$

Regner vi ut de partiellderiverte til L, og setter disse uttrykkene lik null, får førsteordens optimalitetskriterier:

$$\begin{array}{lll} \frac{\partial L}{\partial x_j} = c_j + \mu \frac{1}{x_j} - \sum_i y_i a_{ij} & = 0, & j = 1, 2, \ldots, n, \\ \frac{\partial L}{\partial w_i} = \mu \frac{1}{w_i} - y_i & = 0, & i = 1, 2, \ldots, m, \\ \frac{\partial L}{\partial y_i} = b_i - \sum_j a_{ij} x_j - w_i & = 0, & i = 1, 2, \ldots, m. \end{array}$$

På matriseform

$$\begin{aligned} A^T y - \mu X^{-1} e &= c \\ y &= \mu W^{-1} e \\ Ax + w &= b, \end{aligned}$$

hvor X og W er diagonale matriser hvor diagonale elementer er henholdsvis komponenter fra x og w, og e er en vektor hvor alle elementene er lik 1. La

$z = \mu X^{-1}e$ være en vektor, og gange denne og andre likning henholdsvis med X og W. Da er førsteordens optimalitetskriterier på formen:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned} \tag{4.2}$$

Legg merke til at første likningen refererer til (P) problem og andre til (D) problem. Skriver tredje og fjerde likningen komponentvis og får

$$\begin{aligned} x_j z_j &= \mu \quad j = 1, 2, \ldots, n \\ y_i w_i &= \mu \quad i = 1, 2, \ldots, m. \end{aligned}$$

Ser at disse to likningene refererer til komplementaritet som vi har sett før. Hvis $\mu = 0$, får vi vanlig komplementæritetskriterier som tilfredstiller optimatitet. Derfor kalles disse to siste likningene *μ-komplementæriterskriterier*. Vi har fått tilsammen $2n + 2m$ likninger med $2n + 2m$ ukjente. Hvis disse likningene er lineære, så kan man anvende Gauss-eliminasjon til å løse dem, men dessverre, de er ikkelineære, og dette gjør at LP er ikketriviell.

4.3.1 Eksistens.

Følgende teorem sier når løsningen til barriere problem eksisterer:

Teorem 4.2. *Det fins en løsning til et barriereproblem hvis og bare hvis primaltillatt og dualtillatt mengde ikke er tom.*

Som resulteres i følgende:

Korollar 4.3. *Hvis primaltillatt mengde ikke er tom og begrenset, så fins for hver $\mu > 0$ en unik løsning*

$(x_\mu, w_\mu, y_\mu, z_\mu)$

til (4.2).

Veien $\{(x_\mu, w_\mu, y_\mu, z_\mu) \colon \mu > 0\}$ kalles *primal-dual central path*. Denne spiller en fundamental rolle for indreproduktsmetoder.
Når μ går mot null, konvergerer central path til en optimal løsning for både primalt og dualt problem. PF-metoden er definert som en iterativ prosess som ved hver iterasjon estimerer en verdi av μ som representerer et punkt på central path som er i en viss forstand nærmere den optimale løsning.

4.4 PF-metoden.

PF-metoden er En Fase-metode, og med dette menes at metoden kan begynne fra et punkt som enten er primal- eller dualtillatt, og gå fra dette direkte til en optimal løsning. Vi starter med vilkårlig valgt strengt positive verdier for alle primale og duale verdier, dvs. $(x, w, y, z) > 0$, og disse verdiene oppdateres iterativt ved bruk av følgende algoritme:

Algorithm 2 Barriere metode.

Estimer en passende verdi for μ.
while ikke optimal **do**
1. Beregn skrittretning $(\Delta x, \Delta w, \Delta y, \Delta z)$ som peker tilnærmet på punktet $(x_\mu, w_\mu, y_\mu, z_\mu)$ på central path.
2. Beregn skrittlengdeparameter θ slik at et nytt punkt

$$\begin{aligned} \tilde{x} &= x + \theta\Delta x, \quad & \tilde{y} &= y + \theta\Delta y, \\ \tilde{w} &= w + \theta\Delta w, \quad & \tilde{z} &= z + \theta\Delta z \end{aligned}$$

fortsetter å ha stengt positive komponeneter.
3. Erstatt (x, w, y, z) med en ny løsning $(\tilde{x}, \tilde{w}, \tilde{y}, \tilde{z})$.
end while

4.4.1 Beregning av skrittretning.

Vårt mål er å finne $(\Delta x, \Delta w, \Delta y, \Delta z)$ slik at det nye punktet $(x+\Delta x, w+\Delta w, y+\Delta y, z+\Delta z)$ ligger tilnærmet på primal-dual central path ved $(x_\mu, w_\mu, y_\mu, z_\mu)$. Betrakt likningene for dette punktet:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e. \end{aligned}$$

Vi ser at det nye punktet $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$ hvis det skulle ligge eksakt på central path, definert ved

$$\begin{aligned} A(x + \Delta x) + (w + \Delta w) &= b \\ A^T(y + \Delta y) - (z + \Delta z) &= c \\ (X + \Delta X)(Z + \Delta Z)e &= \mu e \\ (Y + \Delta Y)(W + \Delta W)e &= \mu e. \end{aligned}$$

Betrakter $(\Delta x, \Delta w, \Delta y, \Delta z)$ som ukjente og omdefinerer likningene over:

$$\begin{aligned} A\Delta x + \Delta w &= b - Ax - w &&=: \rho \\ A^T\Delta y - \Delta z &= c - A^T y + z &&=: \sigma \\ Z\Delta x + X\Delta z + \Delta X\Delta Ze &= \mu e - XZe \\ W\Delta y + Y\Delta w + \Delta Y\Delta We &= \mu e - YWe. \end{aligned}$$

De to vektorene ρ og σ representeter henholdsvis primaltillatt og dualtillatt. For å gjøre likningene lineære, dropper vi den ikkelineære delen og kommer frem til følgende lineært system:

$$\begin{aligned} A\Delta x + \Delta w &= \rho \\ A^T\Delta y - \Delta z &= \sigma \\ Z\Delta x + X\Delta z &= \mu e - XZe \\ W\Delta y + Y\Delta w &= \mu e - YWe. \end{aligned}$$

Vi får et lineært likningssystem med $2n + 2m$ likninger med $2n + 2m$ ukjente. Dette systemet er ikkesingulær (under antagelsen om at A har full rang) og

derfor har det en unik løsning som definerer skrittretningen til PF-metoden. Å droppe ikkelineære delen er den vanligste tilnærmingen til å løse lineære likningssystemer. Metoden kalles *Newton's metode* og beskrives kort i neste kapitlet.

4.4.2 Newton's metode.

Gitt en funksjon

$$F(\xi) = \begin{bmatrix} F_1(\xi) \\ F_2(\xi) \\ \vdots \\ F_N(\xi) \end{bmatrix}, \quad \xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix},$$

fra $\mathbb{R}^N$ til $\mathbb{R}^N$. Newton's metode er en effektiv metode til å finne gode tilnærmelser til $\xi^* \in \mathbb{R}^N$ slik at $F(\xi^*) = 0$. Slike punkter kaller vi nullpunkter eller *røtter av F*.
Gitt et punkt $\xi \in \mathbb{R}^N$, hensikten er å finne skrittretning $\Delta\xi$ slik at $F(\xi+\Delta\xi) = 0$. For F som er ikkelineær, er det umulig å finne $\Delta\xi$, derfor ser vi på de to første leddene i Taylor-polynomet til F

$$F(\xi + \Delta\xi) \approx F(\xi) + F^{'}(\xi)\Delta\xi, \tag{4.3}$$

hvor

$$F^{'}(\xi) = \begin{bmatrix} \frac{\partial F_1}{\partial \xi_1} & \frac{\partial F_1}{\partial \xi_2} & \cdots & \frac{\partial F_1}{\partial \xi_N} \\ \frac{\partial F_2}{\partial \xi_1} & \frac{\partial F_2}{\partial \xi_2} & \cdots & \frac{\partial F_2}{\partial \xi_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_N}{\partial \xi_1} & \frac{\partial F_N}{\partial \xi_2} & \cdots & \frac{\partial F_N}{\partial \xi_N} \end{bmatrix}.$$

Istedenfor å løse den kompliserte likningen $F(\xi + \Delta\xi) = 0$, er det mye enklere å løse likningen

$$F(\xi) + F^{'}(\xi)\Delta\xi = 0. \tag{4.4}$$

Dette er et lineært likningssystem, og hvis Jacobi-matrisen $F^{'}(\xi)$ er invertibel, så har vi:

$$\Delta\xi = -\frac{F(\xi)}{F'(\xi)}. \tag{4.5}$$

Når vi har funnet $\Delta\xi$, kan vi oppdatere $\xi \leftarrow \xi + \Delta\xi$. Prosessen oppdateres helt til vi finner ξ slik at $F(\xi) = 0$.
La oss se på det opprinnelige problemet hvor vi ønsket å finne et punkt på central path. La

$$\xi = \begin{bmatrix} x \\ w \\ y \\ z \end{bmatrix}$$

og

$$F(\xi) = \begin{bmatrix} Ax + w - b \\ A^T y - z - c \\ XZe - \mu e \\ YWe - \mu e \end{bmatrix}.$$

Vi får at

$$F^{'}(\xi) = \begin{bmatrix} A & I & 0 & 0 \\ 0 & 0 & A^T & -I \\ Z & 0 & 0 & X \\ 0 & Y & W & 0 \end{bmatrix},$$

og

$$\Delta\xi = \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \\ \Delta z \end{bmatrix}.$$

4.4.3 Beregning av en tilnærmet verdi for barriereparameter μ.

Vi må se hvordan μ skal velges. Hvis μ velges for liten (dvs., nær 1), øker hver iterasjon med en liten faktor. For små μ forventer man et mindre antall step per iteration. For store μ konvergerer sekvensen til den analytiske midten av tillatte mengden.
Hvis det er gitt et punkt (x, w, y, z), som nesten sikkert ligger på central path, så kan vi ved hjelp av følgende formelen finne μ (se for mer [Vand08]):

$$\mu = \delta \frac{z^T x + y^T w}{n + m}, \tag{4.6}$$

hvor δ er et tall mellom 0 og 1. Denne formelen kan vi bruke til å beregne μ selv om punktet (x, w, y, z) ikke ligger på central path.

4.4.4 Valg av skrittlengdeparameter.

Når vi velger skrittlengdeparameteren θ, må vi passe på at

$$x_j + \theta \Delta x_j > 0, \quad j = 1, 2, \ldots, n.$$

Etter litt regning får vi at

$$\frac{1}{\theta} > -\frac{\Delta x_j}{x_j}, \quad j = 1, 2, \ldots, n.$$

Likeledes skal ulikheten være oppfylt for w, y og z variabler også. Sammen skal den største verdien for θ være gitt ved:

$$\frac{1}{\theta} = \max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\}.$$

Dette valget av θ garanterer ikke streng ulikhet, så vi må ha en parameter r som er nær 1, men strengt mindre enn 1 [8]:

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1.$$

Algorithm 3 PF-algoritme

Bestem $(x, w, y, z) > 0$
while ikke optimal **do**
$\rho = b - Ax - w$
$\sigma = c - A^T y + z$
$\gamma = z^T x + y^T w$
$\mu = \delta \frac{\gamma}{n+m}$
Løs
$A\Delta x + \Delta w = \rho$
$A^T \Delta y - \Delta z = \sigma$
$Z\Delta x + X\Delta z = \mu e - XZe$
$W\Delta y + Y\Delta w = \mu e - YWe$
$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$
$x \leftarrow x + \theta\Delta x, \quad y \leftarrow y + \theta\Delta y,$
$w \leftarrow w + \theta\Delta w, \quad z \leftarrow z + \theta\Delta z$
end while

4.4.5 The Path-Following algorithm.

Oppsummert gitt PF-algoritmen foreslått av [Vand08] under:

Eksempel 2. *Gitt LP problem:*

$$\begin{array}{rll} \textit{maksimer} & 2x_1 - 6x_2 & \\ \textit{f.at} & & \\ & -x_1 - x_2 - x_3 & \leq -2 \\ & 2x_1 - x_2 + x_3 & \leq 1 \\ & x_1, x_2, x_3 & \geq 0 \end{array}$$

Skal finne (x, w, y, z) etter en iterasjon av PF metoden.

Litt forklaring til notasjonsbruk:
x, y, w og z er vektorer med passende størrelse, og X, W, Y og Z er diagonalmatriser med diagonale elementer lik elementene i navngitte vektorer:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \implies X = \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{bmatrix}.$$

e er en vektor med passende størrelse som har elementer kun lik 1, O er en nullmatrise.

[8] Notasjonen $a \wedge b$ betyr det minste tallet av a og b.

La $(x, w, y, z) = (e, e, e, e)$ og bruker $\delta = 1/10$ og $r = 9/10$. Har da

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad w = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$
$$z = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad b = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \quad c = \begin{bmatrix} 2 \\ -6 \end{bmatrix},$$

$$A = \begin{bmatrix} -1 & -1 & -1 \\ 2 & -1 & 1 \end{bmatrix}.$$

Beregner

$$\begin{aligned} \rho &= b - Ax - w = \begin{bmatrix} 0 \\ -2 \end{bmatrix}, & \mu &= \delta \frac{\gamma}{n+m} = \tfrac{1}{10}\frac{5}{5} = \tfrac{1}{10}, \\ \sigma &= c - A^T y + z = \begin{bmatrix} 2 \\ -3 \\ 1 \end{bmatrix}, & k_1 &= \mu e - XZe = \begin{bmatrix} -0,9 \\ -0,9 \\ -0,9 \end{bmatrix}, \\ \gamma &= z^T x + y^T w = 5, & k_2 &= \mu e - YWe = \begin{bmatrix} -0,9 \\ -0,9 \end{bmatrix}. \end{aligned}$$

Får en likning $\tilde{A}\tilde{x} = \tilde{b}$, hvor

$$\tilde{A} = \begin{bmatrix} A & I & O & O \\ O & O & A^T & -I \\ Z & O & O & X \\ O & Y & W & O \end{bmatrix}, \qquad \tilde{x} = \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \\ \Delta z \end{bmatrix}$$
$$\tilde{b} = \begin{bmatrix} \rho \\ \sigma \\ \mu e - XZe \\ \mu e - YWe \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 2 \\ -3 \\ 1 \\ -0,9 \\ -0,9 \\ -0,9 \\ -0,9 \\ -0.9 \end{bmatrix}.$$

Legg merke til at $\tilde{A}$ er en $(2n + 2m) \times (2n + 2m)$ matrise. Løser likningen for $\tilde{x}$:

$$\tilde{x} = (\tilde{A})^{-1}\tilde{b} = \begin{bmatrix} -0,5 \\ -1,4 \\ -0,1333 \\ -2,0333 \\ -2,2667 \\ 1,1333 \\ 1,3667 \\ -0,4 \\ 0,5 \\ -0,7667 \end{bmatrix},$$

hvor

$$\Delta x = \begin{bmatrix} -0,5 \\ -1,4 \\ -0,1333 \end{bmatrix}, \quad \Delta w = \begin{bmatrix} -2,0333 \\ -2,2667 \end{bmatrix},$$
$$\Delta y = \begin{bmatrix} 1,1333 \\ 1,3667 \end{bmatrix}, \quad \Delta z = \begin{bmatrix} -0,4 \\ 0,5 \\ -0,7667 \end{bmatrix}.$$

Regner ut $\theta = \frac{9}{10}(2,2667)^{-1} = 0,3971$. Oppdaterer (x, w, y, z):

$$x = x + \theta\Delta x = \begin{bmatrix} 0,8014 \\ 0,4441 \\ 0,9471 \end{bmatrix}, \quad w = w + \theta\Delta w = \begin{bmatrix} 0,1926 \\ 0,0999 \end{bmatrix},$$
$$y = y + \theta\Delta y = \begin{bmatrix} 1,45 \\ 1,5427 \end{bmatrix}, \quad z = z + \theta\Delta z = \begin{bmatrix} 0,8412 \\ 1,1986 \\ 0,6956 \end{bmatrix}.$$

4.4.6 Fremgangestimering.

Vi har sett tidligere at simpleksmetoden terminerer, men for indrepunktmetoder er situasjonen annerledes. Hver løsning som blir produsert, har kun strengt positive variabler. Dette reiser en del spørsmål. Konvergerer sekvensen av løsninger produsert av PF-metoden? Hvor rask konvergerer metoden? Hvor mange iterasjoner skal metoden ha før grensen for toleranse blir nådd? Men først noen definisjoner. Vi trenger å måle størrelsen av forskjellige vektorer. For hver $1 \leq p \leq \infty$ kan vi definere *p-norm* av en vektor x som:

$$\|x\|_p = \left(\sum_j |x_j|^p \right)^{\frac{1}{p}}.$$

og *sup-norm:*

$$\|x\|_\infty = \max_j |x_j|.$$

Vi trenger et viktig verktøy for estimering, *Hölder's ulikhet*:

$$\begin{aligned} |v^T w| &= |\sum_j v_j w_j| \\ &\leq \sum_j |v_j||w_j| \\ &\leq (\max_j |v_j|)(\sum_j |w_j|) \\ &= \|v\|_\infty \|w\|_1. \end{aligned}$$

Har

$$\begin{aligned} \theta &= r \left(\max_{ij} \left\{ \| \frac{\Delta x_j}{x_j} \|, \| \frac{\Delta w_i}{w_i} \|, \| \frac{\Delta y_i}{y_i} \|, \| \frac{\Delta z_j}{z_j} \| \right\} \right)^{-1} \wedge 1 \\ &= \frac{r}{max(\|X^{-1}\Delta x\|_\infty, \ldots, \|Z^{-1}\Delta z\|_\infty)} \wedge 1. \end{aligned}$$

La $\epsilon > 0$ være en liten toleranse, og la $M < \infty$ være en stor endelig toleranse. Hvis $\|x\|_\infty > M$, da er problemet primal ubegrenset, og algoritmen stopper Hvis $\|y\|_\infty > M$, da er problemet dual ubegrenset. Til slutt, hvis $\|\rho\|_1 < \epsilon$, $\|\sigma\|_1 < \epsilon$ og $\gamma < \epsilon$, så stopper algoritmen, og løsningen er optimal. La $\rho^{(k)}$, $\sigma^{(k)}$, $\gamma^{(k)}$ og $\theta^{(k)}$ være verdier for disse størrelsene i k-te iterasjon. Følgende resultat fra ([Vand08]) om generell utførelse av PF-algoritmen:

Teorem 4.4. *Anta at det fins reelle tall $t > 0$, $M < \infty$ og et heltall K slik at for $\forall k \leq K$*

$$\begin{aligned} \theta^{(k)} &\geq t, \\ \|x^k\|_\infty &\leq M, \\ \|y^k\|_\infty &\leq M. \end{aligned}$$

Da fins en konstant $\bar{M} < \infty$ slik at

$$\begin{aligned} \|\rho^k\|_1 &\leq (1-t)^k \|\rho^{(0)}\|_1, \\ \|\sigma^k\|_1 &\leq (1-t)^k \|\sigma^{(0)}\|_1, \\ \gamma^{(k)} &\leq (1-\tilde{t})^k \bar{M}, \end{aligned}$$

for $\forall k \leq K$ hvor

$$\tilde{t} = t(1-\delta).$$

Teoremet viser bare en delvis konvergens, fordi det avhenger av forutsetningen at skrittlengde ikke blir null. For å vise at skrittlengde faktisk har denne egenskapen, kreves det at algoritmen bli modifisert, og at startpunktet skal være nøye utvalgt. Primal og dual ikketillatt går ned med faktor på $1-t$ ved hver iterasjon, mens dualitets gapet går ned saktere med faktor $1-\tilde{t}$.

4.5 Karush-Kuhn-Tucker system.

Den mest tidskrevende operasjon i PF-metoden er å beregne vektorene Δx, Δy, Δw og Δz ut fra disse likningene:

$$A\Delta x + \Delta w = \rho \tag{4.7}$$
$$A^T\Delta y - \Delta z = \sigma \tag{4.8}$$
$$Z\Delta x + X\Delta z = \mu e - XZe \tag{4.9}$$
$$W\Delta y + Y\Delta w = \mu e - YWe. \tag{4.10}$$

Eller litt strev, kan vi skrive dem som en blokkmatrise:

$$\left[\begin{array}{cc|cc} \text{-XZ}^{-1} & & \text{-I} & \\ & & \text{A} & \text{I} \\ \hline \text{-I} & \text{A}^T & & \\ & \text{I} & & \text{YW}^{-1} \end{array}\right] \left[\begin{array}{c} \Delta z \\ \Delta y \\ \hline \Delta x \\ \Delta w \end{array}\right] = \left[\begin{array}{c} \text{-}\mu Z^{-1}e + x \\ \rho \\ \hline \sigma \\ \mu W^{-1}e - y \end{array}\right]. \tag{4.11}$$

Dette systemet kalles $Karush-Kuhn-Tuckersystem$, kalt KKT system. Det er et symmetrisk lineært system med $2n + 2m$ likninger og $2n + 2m$ ukjente. Det er to etapper i reduksjonen av dette systemet. Etter den første etappen er det gjenværende systemet kalt redusert KKT-system, og etter den andre kalles systemet av normale likninger. Vi studerer disse to systemene i de neste kapitlene.

4.5.1 Redusert KKT system.

Løser vi likningene (4.9) og (4.10) for Δz og Δw, og setter disse formlene inn i (4.7) og (4.8), så får vi så kalt *redusert KKT system*:

$$A\Delta x - Y^{-1}W\Delta y = \rho - \mu Y^{-1}e + w \tag{4.12}$$
$$A^T\Delta y + X^{-1}Z\Delta x = \sigma + \mu X^{-1}e - z. \tag{4.13}$$

Skriver systemet i matriseform og får:

$$\begin{bmatrix} -Y^{-1}W & A \\ A^T & X^{-1}Z \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} b - Ax - \mu Y^{-1}e \\ c - A^Ty + \mu X^{-1}e \end{bmatrix}.$$

Vi kan se at redusert KKT matrise er igjen symmetrisk.

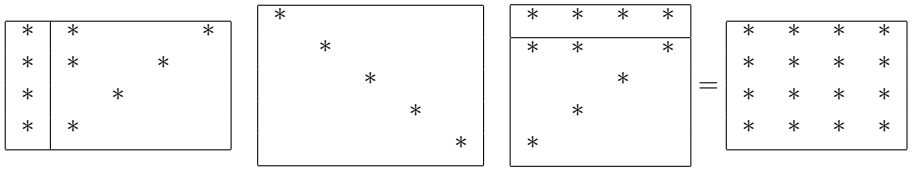
4.5.2 Normallikninger.

Fortsetter å redusere matrisen, og løser (4.13) for Δx og eliminerer Δx fra (4.12) for å få:

$$-(Y^{-1}W + AXZ^{-1}A^T)\Delta y = b - Ax - \mu Y^{-1}e \tag{4.14}$$
$$- AXZ^{-1}(c - A^Ty + \mu X^{-1}e). \tag{4.15}$$

Dette systemet har m likninger og m ukjente, og kalles *normallikninger i primal form*. Dette likningssystemet inneholder matrisen $Y^{-1}W + AXZ^{-1}A^T$, hvor $Y^{-1}W$ er en enkel diagonal matrise (siden W og Y er diagonale matriser). Systemet domineres av $AXZ^{-1}A^T$ matrisen. Hvis A er en sparse matrise(dvs, de fleste elementene er lik null), men har en kolonne med kun ikkenull elementer,

så er $AXZ^{-1}A^T$ lik:

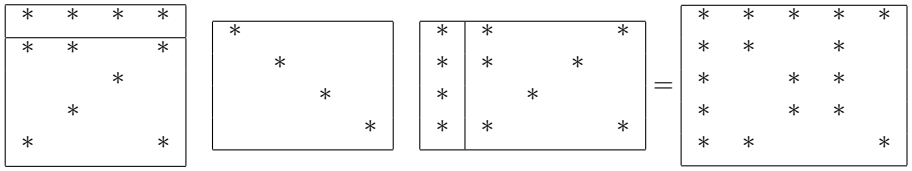


Men vi kan løse likningen (4.12) for Δy og eliminere den fra (4.13):

$$-(A^TYW^{-1} + X^{-1}Z)\Delta x = c - A^Ty + \mu X^{-1}e \tag{4.16}$$

$$+ A^TYW^{-1}(b - Ax - \mu Y^{-1}e). \tag{4.17}$$

Dette systemet har n likninger og n ukjente, og kalles *duale normallikninger*. Nå er kolonner med kun ikkenull elementer, skaper ikke noen problemer:



Sparsitet er alltid viktig for matrisens dimensjoner. For eks, i følge R.Vanderbei, matriser med kun ikkenull elementer forbundet med primale normallikninger krever 65 aritmetiske operasjoner, mens spasematriser med duale normallikninger krever 60 operasjoner. Det gjør ikke så stor forskjell for små matriser, men for store matriser kreves n^3 operasjoner for tette matriser og n operasjoner for sparse. Men det er vanskelig å avgjøre hvilke likninger, primale eller duale som er best å bruke. Det er imidlertid å foretrekke å bruke primale normallikninger hvis sparsematrise ikke har en full kolonne. På samme måte kan det gå galt når vi bruker duale normallikninger hvis matrisen har en full rad. Av den grunn er det best å bruke redusert KKT system direkte.

4.6 Implementasjonsproblemer.

I dette kapittelet vil vi se på gjennomføringsproblemer i forbindelse med PF-metoden. Det viktigste problemet er effektivitet ved bruk av de likningsystemene vi så på i forrige seksjonen. Det er tre valg vi har, vi kan enten bruke redusert KKT matrise:

$$B = \begin{bmatrix} -E^{-2} & A \\ A^T & D^{-2} \end{bmatrix}, \tag{4.18}$$

eller en av to matriser

$$AD^2A^T + E^{-2} \tag{4.19}$$

eller

$$A^TE^2A + D^{-2}. \tag{4.20}$$

(Her $E^{-2} = Y^{-1}W$ og $D^{-2} = X^{-1}Z$.)

Grunnen til at vi vil se på systemet med normallikninger er at denne matrisen er positiv definitt. Å jobbe med positiv definitte matriser har nemlig viktige fordeler.

4.6.1 Faktorisering av positiv definitte matriser.

Matrisene (4.19) og (4.20) er positiv semidefinitte, og ikke nok med det, de er positiv definitte. En matrise B er *positiv semidefinitt* hvis $\xi^T B\xi \geq 0$ for alle vektorer ξ. Summen av to positiv semidefinitte matriser er positiv semidefinitt, og inverse av en symmetrisk positiv semidefinitt matrise er positiv semidefinitt. En matrise B er positiv definitt hvis $\xi^T B\xi > 0$ for alle vektorer $\xi \neq 0$.
Hvis vi begrenser rad/kolonne endringene til symmetriske permutasjoner [Vand08], så er det ingen fare å møte pivotelement der verdien er null. Derfor kan rad/kolonne permutasjon velges på forhånd ved ønske om å opprettholde sparsitet. Hvis vi begrenser oss til symmetriske permutasjoner, hver pivotelement er et diagonal element i matrisen. Følgende resultat viser at vi kan starte med å plukke et tilfeldig diagonal element som første pivot element:

Teorem 4.5. *Hvis B er en positiv definitt matrise, så $b_{ii} > 0$ for $\forall$ i.*

Gitt

$$B = \begin{bmatrix} a & b^T \\ b & C \end{bmatrix}, \tag{4.21}$$

ett trinn av eliminasjonen transformerer B til

$$\text{B=}\begin{bmatrix} a & b^T \\ b & \text{C -}\frac{bb^T}{a} \end{bmatrix}.$$

Neste teoremet fra Vanderbei forteller at denne uteliminerte delen er positiv definitt:

Teorem 4.6. *Hvis B er en positiv definitt matrise, så er $C - bb^T/a$ det.*

Det følger av induksjon at hver uteliminert del er positiv definitt.
Arbeidet i hvert trinn av indrepunktsmetoden er dominert av kravet om å løse likniingssystemet (4.14). Dette bruker nesten 90% av beregningstiden [Mars90]. Likningssystemet kan løses på forskjellige måter, og en av dem er *Cholesky factorisering* $B = R^T R$ hvor R er en *øvre triangular* matrise. Gitt likningen $Bx = b$, kan den reduseres til

$$(R^T R)x = b, \tag{4.22}$$

som kan løses ved å regne

$$R^T r = b, \tag{4.23}$$

for r *(forward substitution)*, og videre løse

$$Rx = r, \tag{4.24}$$

for x *(backward substitution)*.

4.6.2 Kvasidefinitte matriser.

Vi skal se på faktoriseringsteknikker for redusert KKT matrise. Redusert KKT matrise er et eksempel på kvasidefinitte matriser. En symmetrisk matrise kalles *kvasidefinitt* hvis den kan skrives som

$$B = \begin{bmatrix} -E & A \\ A^T & D \end{bmatrix}, \tag{4.25}$$

hvor E og D er positiv definitte matriser. Kvasidefinitte matriser kan LDL^T faktoriseres som positiv definitte. Gitt denne faktoriseringen kan vi bruke forlengs- og baklengssubstitusjon for å løse systemet. Matematisk sett kan ingen av pivotene bli null. Men de kan bli veldig små og dette kan føre til problemene som vi ikke møter med positiv definitte matriser. Så, selv om kvasidefinite KKT system kan brukes, er de numerisk mindre stabile.

5 Implementasjon av PF-metoden.

Det er laget tre implementasjoner av PF-metoden i Matlab. Den første løser LP problem basert på PF algoritmen fra (4.4.5), se vedlegg A. Den andre, kalt PF_KKT_redusert, bygger på redusert KKT system direkte (se vedlegg B), mens den siste reduserer KKT system til normale likninger(vedlegg C). Siden matrisen vi får er positiv definitt, så kan vi anvende Cholesky faktorisering (vedlegg I). Slik unngår vi å løse den $2m + 2n \times 2m + 2n$ likningssystemet ved divisjon slik som vi gjør i den første implementeringen.
Det er også laget en implementasjon av Simpleksalgoritmen (vedlegg D). I alle testproblemer er startpunktet gitt ved $(x_0, w_0, y_0, z_0) = (e, e, e, e)$. For å illustrere anvendelse av algoritmen, la oss vurdere følgende lineær problem:

Eksempel 3 (Oppgave 2.2 [Vand08])**.**

$$\begin{aligned} maksimer \quad & 2x_1 + x_2 \\ f.at \quad & 2x_1 + x_2 \leq 4 \\ & 2x_1 + 3x_2 \leq 3 \\ & 4x_1 + x_2 \leq 5 \\ & x_1 + 5x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Bruker en toleranse $\varepsilon = 10^{-4}$, $M = 1000$, $r = \frac{9}{10}$ og $\delta = \frac{1}{10}$. Starter med$(x, w, y, z) = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)^T$. PF-algoritmen finner optimalløsningen $x^* = (1.0000, 0.0000)$ etter 12 iterasjoner. Metoden bruker 0.000271 (cpu) sek.
PF_KKT_redusert bruker 0.000550 (cpu) sek. Simpleksalgoritmen finner løsningen $x^* = (1, 0)$ etter 1 iterasjon, og bruker 0.000382 (cpu) sekunder.
PF_Normal_Cholesky klarer i løpet av 0.000889 sek . Det er ikke merkbare forskjeller på disse metodene. PF-metoden bruker flere iterasjoner, men dette påvirker ikke regnetiden.

		Simpleks algoritmen		PF-algoritmen	
Nr	Størrelse	Iterasjoner.	CPU tid	Iterasjoner.	CPU tid
1	10×10	2	0.000284	12	0.000639
2	20×20	5	0.000455	15	0.023012
3	30×20	4	0.000907	17	0.053832
4	50×50	2	0.001981	18	0.086937
5	100×100	12	0.001820	26	0.080759
6	300×200	31	0.023055	30	0.592675
7	300×300	22	0.013751	28	1.583721
8	500×500	-	-	37	5.713012
9	1000×700	-	-	38	6.053832
100	1000×1000	-	-	40	10.086937

Tabell 1: Viser antall iterasjoner og cpu tid for Simpleks og PF-algoritmen.

Simpleksalgoritmen er mer effektiv på små LP problemer enn PF-algoritmen. For å utfordre algoritmene må vi teste dem på store LP problemer. Til dette

Nr	Størrelse	Antall iter.	KKT CPU tid	Chol CPU tid
1	10×10	12	0.000748	0.000893
2	20×20	15	0.006216	0.021714
3	30×20	17	0.015393	0.068744
4	50×50	18	0.010053	0.051980
5	100×100	26	0.013049	0.022677
6	300×200	30	0.141272	0.218700
7	300×300	28	0.218737	0.403998
8	500×500	37	1.417300	1.536841
9	1000×700	38	4.927483	4.900142
10	1000×1000	40	9.709345	9.177081

Tabell 2: Viser antall iterasjoner og cpu tid for PF-metoden med redusert KKT system og PF-metoden med Cholesky faktorisering.

formålet kan vi lage en LP-generator som produserer LP problemer av ønsket størrelse ved hjelp av en random funksjon, se vedlegg I. Resultatene fra denne testen vises i tabellene (1-2).
Det er viktig å understreke at LPproblemene som ble testet i denne oppgaven, kan neppe støttes på i virkeligheten siden koeffisientene i objektivfunksjonen og ved begrensningsvariablene er tilfeldig valgt. Det er et viktig moment som skal tas i betrakning ved gjennomgang av testdata. I tillegg har regnekapasiteten til datamaskinen spilt en rolle for størrelsen på LPproblemer som programmet klarte å løse. Simpleksalgoritmen stoppet allerede for 500×500 problemer, mens 1000×1000 var grensen for PF-algoritmer, men da ble Matlab's arbeidsminne oversteget. Testen viser immidlertid at PF-algoritmer som bygger på redusert KKTsystem og Cholesky faktorisering er betraktelig raskere anvendt på store LP problemer enn PF algoritmen som ikke bygger på KKT system eller normallikninger. Legg merke til at antall iterasjoner ved PF algoritmen øker saktere når størrelsen til LP problem økes.

5.1 Forskjellen mellom PF- og Simpleksmetoden.

Simpleksalgoritmen løser et lineær programmeringsproblem ved å flytte langs kantene på en polytope definert av begrensninger, fra hjørne til hjørne med større verdier av objektivfunksjon helt til maksimum er nådd. Resultatet ved gjennomføringen av denne algoritmen er at det er mulig å løse problemer eksakt. Derfor er disse metodene egnet for små problemer hvor det er ønkelig å finne en løsning på toppunktet, eller hvor det er ønskelig å finne en basis.
Indrepunktsalgoritmer for lineær programmering løser problemer fra det indre av polytope definert av begrensninger. De kommer nærmere løsningen svært raskt, men i motsetning til simpleksalgoritmer, finner ikke løsningen eksakt. Indrepunktsmetoden er mer effektiv for store systemer og dermed bør brukes i slike tilfeller.

Eksempel 4. *Gitt et LP problem*

$$\begin{aligned} maksimer \quad & x + y \\ f.at \quad & x + y \quad \leq 1 \\ & x, y \quad \geq 0 \end{aligned}$$

Testen viser at Simpleksalgoritmen og PF-algoritmen gir forskjellige løsninger på problemet (se tabellen (3)). Maksimumet kan ligge i et skjæringspunkt,

Program	Optimal løsning	Antall iterasjoner	CPU tid
Simpleksalg.	(1,0)	1	0.114307
PF	(0.5,0.5)	7	0.000369

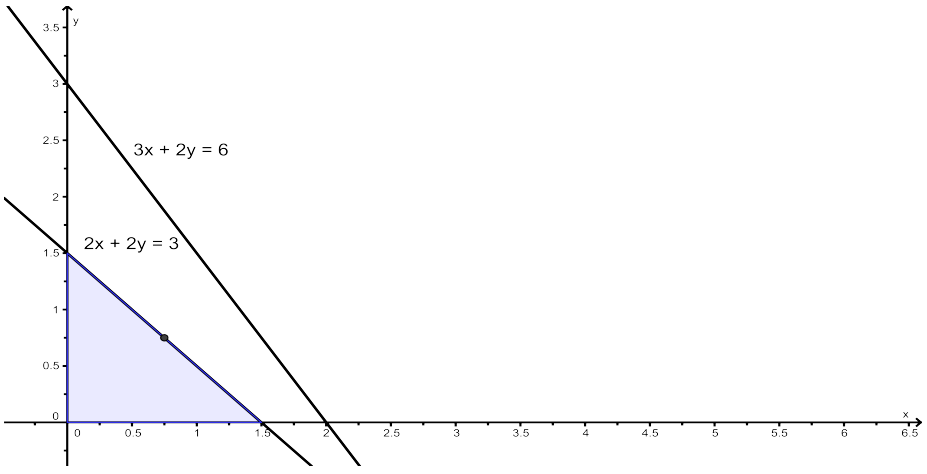
Tabell 3: Tabellen viser en optimalløsning beregnet ved hjelp av simpleksalgoritmen og PF-algoritmen.

eller på en linje mellom to skjæringspunkter. Dersom den optimale verdien ligger på en linje mellom to punkter som er i dette eksempelet, vil alle punktene på linjen være optimale løsninger. Men da er endepunktene til linjen også optimale. Simpleksalgoritmen oppgir uansett en optimal løsning i et skjæringspunkt. Den går gjennom punktene i utfallsrommets ytterkanter og terminerer på et lokalt optimum. Indrepunktsalgoritmen gir derimot en løsning som ligger i midten av løsningsmengden. Vi skal se nærmere på følgende eksempel:

Eksempel 5. *Gitt et LP problem*

$$\begin{aligned} maksimer \quad & x + y & \\ f.at \quad & 3x + 2y & \leq 6 \\ & 2x + 2y & \leq 3 \\ & x, y & \geq 0 \end{aligned}$$

Problemet har flere løsninger. Simpleksalgoritmen finner løsningen i et av hjørnene i polytopen, se figuren (12). Nivåkurven til objektivfunksjonen med



Figur 12: PF-algoritmen finner midtpunktet på en linje mellom to skjæringspunkter.

optimalverdi faller sammen med utfallsrommets kant. De to algoritmene leverer hver sin løsning. Resultatene fra testen vises i tabellen (4).
Vi ser på et nytt problem med flere løsninger.

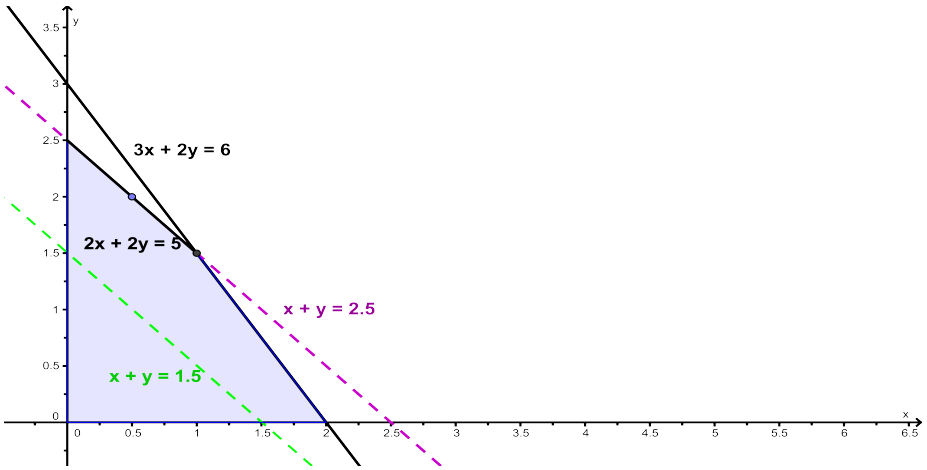
Program	Optimal løsning	Optimal verdi	Antall iterasjoner
Simpleksalg.	(1.5, 0)	1.5	1
PF	(0.7492, 0.7508)	1.5	8

Tabell 4: Løsningen i oppgaven (5).

Eksempel 6. *Gitt et LP problem*

$$\begin{aligned} maksimer \quad & x+y & \\ f.at \quad & 3x+2y & \leq 6 \\ & 2x+2y & \leq 5 \\ & x,y & \geq 0 \end{aligned}$$

En del av nivåkurven til objektivfunksjon faller sammen med en kant i polytopen, se figur (13). PF-algoritmen finner midtpunktet for denne kanten, mens



Figur 13: PF-algoritmen finner midtpunktet på en linje mellom to skjæringspunkter.

Simpleksalgoritmen oppgir løsningen som ligger i hjørnet (tabell (5)). Vi ser til

Program	Optimal løsning	Optimal verdi	Antall iterasjoner
Simpleksalg.	(1, 1.5)	2.5	2
PF	(0.4806, 2.0194)	2.5	8

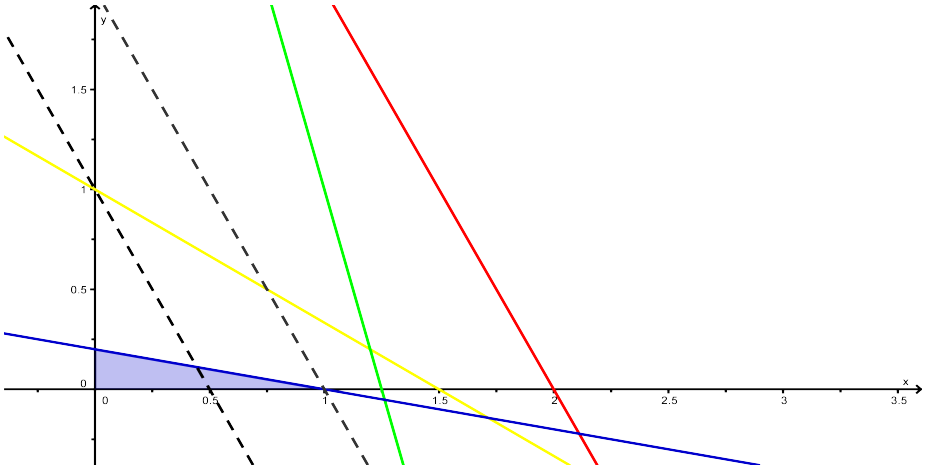
Tabell 5: Forskjellen mellom Simpleks og PF algoritmen.

slutt på et annet LP-problem.

Eksempel 7. *Betrakt et LP problem*

$$\begin{array}{rrl} maksimer & 2x+y & \\ f.at & 2x+y & \leq 4 \\ & 2x+3y & \leq 3 \\ & 4x+y & \leq 5 \\ & x+5y & \leq 1 \\ & x,y & \geq 0 \end{array}$$

Nivåkurvene til objektivfunksjonen er ikke parallelle med noen av kantene til polytopen (figur (14)). Begge algoritmene finner lik løsning, og PF-algoritmen



Figur 14: Nivåkurvene til objektivfunksjonen i eksempel (7).

bruker flere iterasjoner, mens Simpleks klarer med en (tabell (6)).

Program	Optimal løsning	Optimal verdi	Antall iterasjoner
Simpleksalg.	(1, 0)	2	1
PF	(1, 0)	2	14

Tabell 6: Forskjellen mellom Simpleks og PF algoritmen.

6 Anvendelse.

6.1 Matriseapproksimasjon.

Først litt notasjon:
For en gitt matrise $A \in \mathbb{R}^{m\times n}$, lar vi *R(A)* og *C(A)* betegne h.h.v. vektoren bestående av alle radsummer og kolonnesummer i A. Så er $C(A) = (c_1, c_2, \ldots, c_n)$, og $c_j = \sum_{i=1}^{m} a_{ij}$ (for $j \leq n$).
Vi definerer avstandsfunksjonen

$$\text{d(A,B)}= \sum_{ij} \mid a_{ij} - b_{ij} \mid$$

for to matriser $A, B \in \mathbb{R}^{m\times n}$.

Problem 1. *La nå følgende være oppgitt: $B \in \mathbb{R}^{m\times n}$, r^l, $r^u \in \mathbb{R}^m$, og c^l, $c^u \in \mathbb{R}^n$. Betrakt optimeringsproblemet*

$$\min\{d(A,B) : r^l \leq R(A) \leq r^u, \quad c^l \leq C(A) \leq c^u, \quad A \geq O\}$$

der ulikheten holder for hver komponent.

Så vi har et minimeringsproblem:

$$\begin{aligned} \textit{minimer} \quad & \sum_{ij} \mid a_{ij} - b_{ij} \mid \\ \textit{f.at} \quad & \\ & r^l \leq R(A) \leq r^u \\ & c^l \leq C(A) \leq c^u \\ & A \geq O \end{aligned}$$

Problemet er å finne den nærmeste matrisen A til den gitte matrisen B som oppfyller noen betingelser. Dette problemet kan konverteres til et LP problem slik at vi kan bruke kjente metoder og teknikker til å løse det:

$$\begin{aligned} \textit{minimer} \quad & \sum_{ij} z_{ij} \\ \textit{f.at} \quad & \\ r_i^l \leq \; & \textstyle\sum_{j=1}^{m} a_{ij} \leq r_i^u \quad i = 1, \ldots, m \\ c_j^l \leq \; & \textstyle\sum_{i=1}^{m} a_{ij} \leq c_j^u \quad j = 1, \ldots, n \\ & a_{ij} - b_{ij} \leq z_{ij} \\ & -a_{ij} + b_{ij} \leq z_{ij} \\ & A \geq O. \end{aligned}$$

I enhver tillatt løsning er $\mid a_{ij} - b_{ij} \mid \leq z_{ij}$, og i enhver optimal løsning må $z_{ij} = \mid a_{ij} - b_{ij} \mid$. Videre får vi:

$$\begin{aligned} \textit{minimer} \quad & \sum_{ij} z_{ij} \\ \textit{f.at} \quad & \\ r_i^l \leq \; & \textstyle\sum_{j=1}^{m} a_{ij} \leq r_i^u \quad i = 1, \ldots, m \\ c_j^l \leq \; & \textstyle\sum_{i=1}^{m} a_{ij} \leq c_j^u \quad j = 1, \ldots, n \\ & a_{ij} - z_{ij} \leq b_{ij} \\ & -a_{ij} - z_{ij} \leq -b_{ij} \\ & A \geq O. \end{aligned}$$

$$
\begin{aligned}
\textit{minimer} \quad & \sum_{ij} z_{ij} \\
\textit{f.at} \quad & \\
& \textstyle\sum_{j=1}^{m} a_{ij} \leq r_i^u \qquad i = 1, \ldots, m \\
& -\textstyle\sum_{j=1}^{m} a_{ij} \leq -r_i^l \qquad i = 1, \ldots, m \\
& \textstyle\sum_{i=1}^{m} a_{ij} \leq c_j^u \qquad j = 1, \ldots, n \\
& -\textstyle\sum_{i=1}^{m} a_{ij} \leq -c_j^l \qquad j = 1, \ldots, n \\
& a_{ij} - z_{ij} \leq b_{ij} \\
& -a_{ij} - z_{ij} \leq -b_{ij} \\
& A \geq O.
\end{aligned}
$$

Siden matrise A er den ukjente i problemet, kan vi kalle den X isteden. Da er kolonnesummen og radsummen kan skrives slik:

$$
\sum_{j=1}^{n} X_{ij} \in \left[r_i^l, r_i^u\right], \quad i = 1, \ldots, m, \quad \sum_{i=1}^{m} X_{ij} \in \left[c_j^l, c_j^u\right], \quad j = 1, \ldots, n.
$$

Etter den nye notasjonen kan problemet omformuleres slik

$$
\begin{aligned}
\textit{minimer} \quad & \sum_{ij} z_{ij} \\
\textit{f.at} \quad & \\
& \textstyle\sum_{j=1}^{m} X_{ij} \leq r_i^u \qquad i = 1, \ldots, m \\
& -\textstyle\sum_{j=1}^{m} X_{ij} \leq -r_i^l \qquad i = 1, \ldots, m \\
& \textstyle\sum_{i=1}^{m} X_{ij} \leq c_j^u \qquad j = 1, \ldots, n \\
& -\textstyle\sum_{i=1}^{m} X_{ij} \leq -c_j^l \qquad j = 1, \ldots, n \\
& x_{ij} - z_{ij} \leq b_{ij} \\
& -x_{ij} - z_{ij} \leq -b_{ij} \\
& X \geq O.
\end{aligned}
$$

$$
\begin{array}{llllllllllll}
\textit{minimer} & 0\cdot x_{11} & + & \dots & + & 0\cdot x_{mn} & + & z_{11} & + & \dots & + & z_{mn} \\
\textit{f.at} & & & & & & & & & & & \\
 & x_{11} & + & x_{12} & + & \dots & + & x_{1n} & \leq & r_1^u & & \\
 & \vdots & + & \vdots & + & \dots & + & \vdots & \leq & \vdots & & \\
 & x_{m1} & + & x_{m2} & + & \dots & + & x_{mn} & \leq & r_m^u & & \\
 & -x_{11} & - & x_{12} & - & \dots & - & x_{1n} & \leq & -r_1^l & & \\
 & \vdots & - & \vdots & - & \dots & - & \vdots & \leq & \vdots & & \\
 & -x_{m1} & - & x_{m2} & - & \dots & - & x_{mn} & \leq & -r_m^l & & \\
 & x_{11} & + & x_{21} & + & \dots & + & x_{m1} & \leq & c_1^u & & \\
 & \vdots & + & \vdots & + & \dots & + & \vdots & \leq & \vdots & & \\
 & x_{1n} & + & x_{2n} & + & \dots & + & x_{mn} & \leq & c_n^u & & \\
 & -x_{11} & - & x_{21} & - & \dots & - & x_{m1} & \leq & -c_1^l & & \\
 & \vdots & - & \vdots & - & \dots & - & \vdots & \leq & \vdots & & \\
 & -x_{1n} & - & x_{2n} & - & \dots & - & x_{mn} & \leq & -c_n^l & & \\
 & & & & & x_{11} & - & z_{11} & \leq & b_{11} & & \\
 & & & & & \vdots & - & \vdots & \leq & \vdots & & \\
 & & & & & x_{mn} & - & z_{mn} & \leq & b_{mn} & & \\
 & & & & & -x_{11} & - & z_{11} & \leq & -b_{11} & & \\
 & & & & & \vdots & - & \vdots & \leq & \vdots & & \\
 & & & & & -x_{mn} & - & z_{mn} & \leq & -b_{mn} & & \\
 & & & & x_{11}, & x_{21}, & \dots, & x_{mn} & \geq & 0 & &
\end{array}
$$

Videre kan vi bruke *vec* operatoren som oppretter en kolonnevektor fra en ma-

trise A ved å stable kolonnevektorene $A = [a_1, a_2, \ldots, a_n]$ under hverandre:

$$vec(A) = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}.$$

La $x = vec(X)$, $z = vec(Z)$ og $b = vec(B)$, hvor X, Z og $X \in \mathbb{R}^{m \times n}$. Vi vet at et hvert minimeringsproblem kan formuleres som et maksimeringsproblem: $min\ c^t x$ = - max -$c^t x$. Så, det opprinnelige problemet kan nå formuleres som et LP problem på matriseform slik vi vant å se:

$$\begin{aligned} -maksimer \quad & -c^T x^* \\ f.at \quad & \\ & A^* x^* \leq b^* \\ & x \geq 0, \end{aligned}$$

hvor $x^* = \begin{bmatrix} x \\ z \end{bmatrix}$, $c = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$ og $b^* = \begin{bmatrix} r^u \\ -r^l \\ c^u \\ -c^l \\ b \\ -b \end{bmatrix}$.

A^* stilt opp som en blokkmatrise, er på formen:

$$\mathrm{A}^* = \begin{bmatrix} \mathrm{I}_m & \ldots & \mathrm{I}_m & \mathrm{O} \\ \text{-}\mathrm{I}_m & \ldots & \text{-}\mathrm{I}_m & \mathrm{O} \\ & \mathrm{D} & & \mathrm{O} \\ & \text{-}\mathrm{D} & & \mathrm{O} \\ & \mathrm{I}_{mn} & & \mathrm{I}_{mn} \\ & \text{-}\mathrm{I}_{mn} & & \text{-}\mathrm{I}_{mn} \end{bmatrix},$$

hvor I_m er en $m \times m$ identitetsmatrise, O er en nullmatrise og I_{mn} er $mn \times mn$ identitetsmatrise. D matrise er litt spesiell, og er på formen:

$$D = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & \ldots & 1 & 0 & \ldots & 0 \\ & & & \vdots & & \vdots & & & \\ 0 & 0 & 0 & \ldots & 0 & 1 & \ldots & 1 & 1 \end{bmatrix}.$$

Problemet har $2mn$ ukjente og $2(mn + m + n)$ ulikheter, men kan lett løses ved hjelp av PF-algoritmen.

La $B = \begin{bmatrix} 5 & 3 & 2 \\ 1 & 4 & 3 \end{bmatrix}$, $r^u = \begin{bmatrix} 12 \\ 8 \end{bmatrix}$, $r^l = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$, $c^u = \begin{bmatrix} 8 \\ 6 \\ 10 \end{bmatrix}$ og $c^l = \begin{bmatrix} 4 \\ 0 \\ 5 \end{bmatrix}$.
Bruker en toleranse $\varepsilon = 10^{-4}$, $M = 1000$, $r = \frac{9}{10}$ og $\delta = \frac{1}{10}$. Startpunktet er gitt ved $(x_0, w_0, y_0, z_0) = (e, e, e, e)$.
Metoden bruker 0.001526 sek og 13 iterasjoner, og oppgir matrisen

$$X = \begin{bmatrix} 5 & 2.6154 & 2 \\ 1 & 3.3846 & 3 \end{bmatrix}$$

som den beste matriseapproksimasjonen, og avstanden lik 0.7257 avrundet.

6.2 Alternerende projeksjoner.

Vi skal analysere nå en alternativ metode for problem (1), nemlig *alternerende proeksjoner*. Alternerende proeksjoner regnes som en enkel algoritme for å beregne skjæringspunktet mellom konvekse mengder ved bruk av sekvens av projeksjoner på disse mengdene.
Anta at C og D er to lukkede konvekse mengder i $\mathbb{R}^n$, og la P_C og P_D være projeksjoner in i C og D henholdsvis. Algoritmen starter fra et vilkårlig punkt $x_0 \in C$, og projiserer vekselvis inn i C og D:

$$y_k = P_D(x_k), \quad x_{k+1} = P_C(y_k), \quad k = 0, 1, 2, \ldots$$

Disse projeksjonene genererer sekvens av punkter $x_k \in C$ og $y_k \in D$. Følgende resultat som knyttes til Cheney og Goldstein (les mer i [ChGold59]), gjengitt av William K.Glunt [Glu94], og regnes som veldig viktig for projeksjoner:

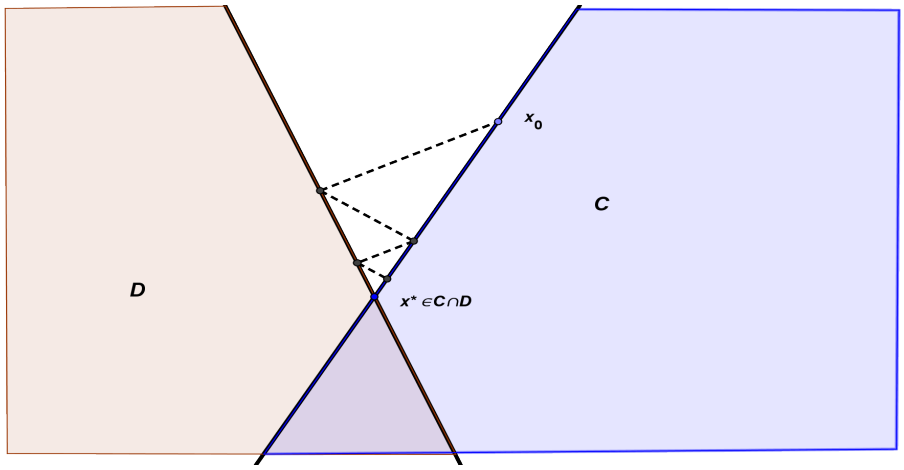
Teorem 6.1. *La C og D være stengt konvekse mengder i et Hilbert rom H. Hvis mengdene er enten kompakte eller endelig dimensjonale, og hvis avstanden mellom de to mengdene er oppnåelig, så for vilkårlig $x_0 \in H$ konvergerer sekvensene generert av $y_{k+1} = P_D(x_k)$ og $x_{k+1} = P_C(y_{k+1})$ mot punkter x og y slik at*

$$\|x - y\| = \inf_{r \in C, s \in D} \|r - s\|.$$

Hvis $C \cap D \neq \varnothing$, så konvergerer begge sekvensene x_k og y_k til et punkt $x^* \in C \cap D$. Med andre ord, alternerende projeksjoner finner skjæringspunktet av mengder.
La nå $C \cap D = \varnothing$, og anta at det fins punkter i C og D slik at avstanden mellom C og D er *dist(C,D)*. Da er $x_k \to x^* \in C$, og $y_k \to y^* \in D$, slik at $\|x^* - y^*\|_2 = dist(C, D)$. Så, alternerende projeksjoner finner i dette tilfellet et par punkter i C og D slik at avstanden mellom dem er minimal.

6.2.1 Konvergens av alternerende projeksjoner.

La $C \cap D \neq \varnothing$. Beviset for konvergens for alternerende projeksjoner baserer seg på rapporten av Boyd og Dattorro [BDat03], men kan finnes i andre rapporter, bl.a. av W. Glunt, W. Cheney og A. Goldstein.



Figur 15: Etter noen få iterasjoner konvergerer metoden rask til $x^* \in C \cap D$.

La $x' = C \cap D$. Har at

$$\begin{aligned} \|x_k - x'\|^2 &= \|x_k - y_k + y_k - x'\|^2 \\ &= \|x_k - y_k\|^2 + \|y_k - x'\|^2 + 2(x_k - y_k)^T(y_k - x') \\ &\geq \|x_k - y_k\|^2 + \|y_k - x'\|^2. \end{aligned}$$

Vi får da

$$\|y_k - x'\|^2 \leq \|x_k - x'\|^2 - \|y_k - x_k\|^2.$$

Dette viser at y_k er nærmere x' enn x_k er. Lignende kan vi vise at

$$\|x_{k+1} - x'\|^2 \leq \|y_k - x'\|^2 - \|x_{k+1} - y_k\|^2,$$

dvs, x_{k+1} er nærmere x' enn y_k er. Observerer først at

$$\|x_k - x'\| \leq \|x_0 - x'\|, \quad \|y_k - x'\| \leq \|x_0 - x'\|, \quad k = 1, 2, \ldots$$

Vi kan konkludere med at sekvensene x_k og y_k er begrenset. Derfor har sekvensen x_k et akkumulasjonspunkt x^*. Siden C er lukket, og $x_k \in C$, så har vi at $x^* \in C$. Vi skal vise at $x^* \in D$, og at sekvensene x_k og y_k konvergerer begge til x^*. Fra utregningene over får vi at sekvensene

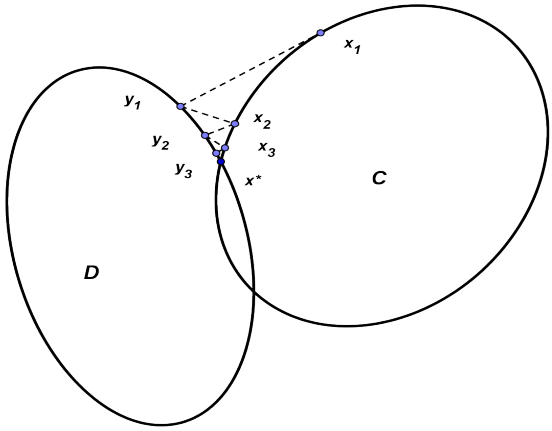
$$\|x_0 - x'\|, \|y_0 - x'\|, \|x_1 - x'\|, \|y_1 - x'\|, \ldots$$

er avtagende, og dermed konvergerer.

Vi kan konkludere nå at $\|y_k - x_k\|$ og $\|x_{k+1} - y_k\|$ må konvergere til 0. Fra

$$dist(x_k, D) = dist(x_k, y_k) \to 0$$

og at D er lukket, kan vi konkludere nå at $x^* \in D$. Dermed, $x^* \in C \cap D$. Siden x^* ligger i skjæringen mellom C og D, så har vi at $x' = x^*$ (det er fordi x' er det eneste skjæringspunktet). Dette gir at $\|y_k - x^*\|$ og $\|x_k - x^*\|$ konvergerer begge til 0.



Figur 16: Begge sekvensene konvergerer til $x^* \in C \cap D$.

6.2.2 Alternerende projeksjonsalgoritme.

Ser tilbake på opprinnelige problemet hvor vi skal finne matrisen $X \in \mathbb{R}^{m\times n}$ hvor kolonnesummen og radsummen har begrensninger

$$\sum_{j=1}^{n} X_{ij} \in \left[r_i^l, r_i^u\right], \quad i = 1, \ldots, m, \quad \sum_{i=1}^{m} X_{ij} \in \left[c_j^l, c_j^u\right], \quad j = 1, \ldots, n.$$

Vi kan vurdere dette problemet definert for C_i, $i \in S = (1, ..., s)$, som er lukkede, konvekse mengder, hvor vi ønker å finne et punkt $x \in C = \cap_{i\in S} C_i$. Den geometriske tolkningen er at vi bestemmer projeksjon av en vektor b på m halvplan definert av $a_i^T x \leq b_i$. Gitt en vektor v, proeksjonen av v på mengden $\left\{z \mid l \leq 1^T z \leq u\right\}$ er

$$P(z) = \begin{cases} z, & l \leq 1^T z \leq u \\ z - ((1^T z - u)/n)1, & 1^T z > u \\ z - ((l - 1^T z)/n)1, & 1^T z < l. \end{cases}$$

En enkel iterasjon starter med et punkt $x_0 \in C$. Vi bestemmer også stoppekriterier, en toleranse ϵ, og projiserer matrisen på hver av mengdene definert av

Algorithm 4 Alternerende projeksjonsalgoritme.

Gitt x_0, ϵ
while $\|x_{k+1} - x_k\| \leq \epsilon$ **do**
  $y_k = P_D(x_k)$
  $x_{k+1} = P_C(y_k)$
  $x^* = x_k$
end while

kolonnesum og radsum.

Det ble gjort en del tester med forskjellige projeksjonsalgoritmer, bl.a av N.Gould (se rapporten [Gou02]). Det er klart umiddelbart for disse testene at indrepunktsmetoden er langt bedre enn alle fire projeksjonsmetoder beskrevet i denne rapporten. Nærmere gjennomgang av de detaljerte resultatene indikerer at selv om projeksjonsmetodene er enklere, er indrepunktsmetoden likevel en klar vinner. Gould ønsker ikke å fastslå at projeksjonsmetoder ikke er nyttige, da de særlig synes å ha vært brukt med hell i mange år i medisinsk bildebehandling, strålebehandling, planlegging og signalbehandling. Men erfaring tilsier at til tross for den store litteraturen viet til teoretisk analyse, bør de ikke anses som metode for valg av et gitt program uten ytterligere sterke empiriske bevis som støtter et slikt krav.

A PF-algoritme

Program 1 PF-algoritme.

```
function optimal_solution = PF_enkel(A,b,c,r,delta)
r=9/10;delta=1/10;
[m,n]=size(A);
x=ones(n,1); z=x; w=ones(m,1); y=w;
epsilon = 0.0001;M=1000;
iter=0;
while 1==1
    iter = iter + 1;
rho= b-A*x-w; sigma=c-A' *y+z;gamma = z'*x+y'*w;
mu = delta*(gamma/(n+m));
X=diag(x);Y=diag(y); Z=diag(z);W=diag(w);
ej=ones(n,1); ei=ones(m,1);
b3=mu*ej-X*Z*ej;b4=mu*ei-Y*W*ei;Ii =eye(m);Ij=-eye(n);
Oi=zeros(m,m); Oij=zeros(m,n);
Ojj=zeros(n,n);Oji=zeros(n,m);
F=[A Ii Oi Oij;Ojj Oji A' Ij;Z Oji Oji X;Oij Y W Oij];
bny=[rho; sigma; b3;b4];
start=tic;
deltax=inv(F)*bny;indexx=[x;w;y;z];
for i = 1:(2*m+2*n)
  T= -deltax(i)/indexx(i);maks(i)=T;
end
[maks i]=max(maks);maks=inv(maks);teta=r*maks;
if teta <1
teta =teta;
else teta =1;
end
x=x+teta*deltax(1:n);w=w+teta*deltax(n+1:n+m);
y=y+teta*deltax(n+m+1:n+m+m);z=z+teta*deltax(n+2*m+1:2*n+2*m);
% Stoppe kriterier
P=1;nx=norm(x,inf);ny=norm(y,inf);nrho = norm(rho,P);nsigma= norm(sigma,P);
if nx > M
    fprintf('Problemet er primal ubegrenset.')
   break
   toc(start)
end
```

```
if ny > M
    fprintf('Problemet er dual ubegrenset og dermed ikke er primaltillatt.')
toc(start)
break
end
if nrho < epsilon && nsigma < epsilon && gamma < epsilon
 optimal_solution = x
antall_iterasjoner = iter
toc(start)
    return
end
end
```

B PF-algoritme med redusert KKT system.

Program 2 PF-algoritme redusert KKT system.

```
function optimal_solution = PF_KKT_redusert(A,b,c,r,delta)
r=9/10;delta=1/10;[m,n]=size(A);
x=ones(n,1); z=x; w=ones(m,1); y=w;
epsilon = 0.0001; M=1000;trinn=0;
while 1==1
    trinn = trinn + 1;
rho= b-A*x-w; sigma=c-A' *y+z;gamma = z'*x+y'*w;
mu = delta*(gamma/(n+m));X=diag(x);
Y=diag(y); Z=diag(z);W=diag(w);
ej=ones(n,1); ei=ones(m,1);b1 = b-A*x-mu*inv(Y)*ei;
b2 = c-A'*y+mu*inv(X)*ej;F=[-inv(Y)*W A; A' inv(X)*Z];bny=[b1;b2];
start=tic;
dyx=inv(F)*bny;dx = dyx(m+1:n+m);dy=dyx(1:m);
dz = inv(X)*(mu*ej-X*Z*ej-Z*dx);dw = inv(Y)*(mu*ei-Y*W*ei-W*dy);
dxwyz = [dx;dw;dy;dz];indexx=[x;w;y;z];
for i = 1:(2*m+2*n)
  Tny= -dxwyz(i)/indexx(i); maks(i)=Tny;
end
[maks i]=max(maks);maks=inv(maks);teta=r*maks;
if teta <1
teta =teta;
else teta =1;
end
x=x+teta*dx;w=w+teta*dw;y=y+teta*dy;z=z+teta*dz;
% Stoppe kriterier
P=1;gamma=gamma;
nx=norm(x,inf); ny=norm(y,inf);nrho = norm(rho,P);nsigma= norm(sigma,P);
if nx > M
       antall_trinn = trinn
    fprintf('Problemet er primal ubegrenset.')
   return
   toc(start)
end
```

```
if ny > M
       antall_trinn = trinn
   fprintf('Problemet er dual ubegrenset og dermed ikke er primaltillatt.')
toc(start)
return
end
if nrho < epsilon && nsigma < epsilon && gamma < epsilon
      antall_iterasjoner = trinn
 optimal_solution = x
 primal_obj = dotprod(c',x,n)
dual_obj = dotprod(b',y,m)
toc(start)
    return
end
end
```

C PF-algoritme ved bruk av Cholesky faktoriseringen.

Program 3 PF-algoritmen med Cholesky faktoriseringen.

```
 function optimal_solution = PF_Normal_Cholesky(A,b,c,r,delta)
r=0.9;delta=0.1;[m,n]=size(A);x=ones(n,1); z=x; w=ones(m,1); y=w;
epsilon = 0.0001; M=1000;trinn=0;
while 1==1
start=tic;
rho= b-A*x-w; sigma=c-A' *y+z;gamma = z'*x+y'*w;
mu = delta*(gamma/(n+m));X=diag(x);Y=diag(y); Z=diag(z);W=diag(w);
ej=ones(n,1); ei=ones(m,1);b1 = b-A*x-mu*inv(Y)*ei;
b2 = c-A'*y+mu*inv(X)*ej;F=(inv(Y)*W+A*X*inv(Z)*A');
bny=b1-A*X*inv(Z)*b2; trinn = trinn + 1;
R=Cholesky(F); %kjører cholesky funksjonen som finner R som er upper triangulær
yny=Forwardsolve(R,bny);
dy=Backsolve(R,yny); %denne funksjonen løser upper triangulær system Rx = y
dy=-dy;
dx=X*inv(Z)*(b2-A'*dy);dz = inv(X)*(mu*ej-X*Z*ej-Z*dx);
dw = inv(Y)*(mu*ei-Y*W*ei-W*dy);dxwyz = [dx;dw;dy;dz];indexx=[x;w;y;z];
for i = 1:(2*m+2*n)
  Tny= -dxwyz(i)/indexx(i); maks(i)=Tny;
end
[maks i]=max(maks);maks=inv(maks);teta=r*maks;
if teta <1
teta =teta;
else teta =1;
end
x=x+teta*dx;w=w+teta*dw;y=y+teta*dy;z=z+teta*dz;
% Stoppe kriterier
P=1;nx=norm(x,inf); ny=norm(y,inf);nrho = norm(rho,P);nsigma= norm(sigma,P);
if nx > M
        antall_iterasjoner=trinn
    fprintf('Problemet er primal ubegrenset.')
    toc(start)
   return
end
if ny > M
       antall_iterasjoner=trinn
    fprintf('Problemet er dual ubegrenset, dermed ikke er primaltillatt.')
toc(start)
return
end
```

```
if nrho < epsilon && nsigma < epsilon && gamma < epsilon
   antall_iterasjoner=trinn
 optimal_solution = x
  primal_obj = dotprod(c',x,n)
dual_obj = dotprod(b',y,m)
toc(start)
    return
end
end
```

D Simpleksalgoritme i matriseform.

Program 4 Simpleksalgoritme i matriseform.

```
function optimal_solution = SimpleksToFaseMetode(A,b,c)
[m,n]=size(A); N=A;B=eye(m);A=[N B];xstjerneB = B\b;
cNg=c; cN = c;B_index = n+1:n+m;bg=b;
N_index = 1:n;zstjerneN = -cN;cNtest =c;
% tester om betingelsen for valg av metoden
[test_xstjerneB b]= min(xstjerneB);tell=0;
for k=1:n
    [test_c c]= min(cNtest);
    if test_c <0
        tell=tell+1;
        cNtest(c)= abs(test_c);
    end
end
%_____________________Dual Simpleks_____________________________________
if test_xstjerneB<0 & tell==n
 fprintf ('Starter dual simpleksalgoritme')
start=tic;Dual_Simpleks
toc(start)
end
%_______________________________To-fase Simpleks_____________________
%
 if test_xstjerneB<0  & tell <n
  fprintf ('Starter To-fase simpleksalgoritme')
  cN =-ones(n,1);zstjerneN = -cN;
Dual_Tofase
 end
%%_____________________________________________________________________________
% ------------------------PRIMAL SIMPLEKSALGORITME-------------------------
if test_xstjerneB>=0
  fprintf ('Starter Primal simpleksalgoritme')
Primal_Simpleks
end
end
```

E Dual Simpleksalgoritme.

Program 5 Dual Simpleksalgoritme i matriseform.

```
 % Kjører DUAL SIMPLEKSALGORITME
 % STEP 1 Hvis XstjerneB < 0 så løsningen ikke optimal
pivot = 0;
while 1==1
    pivot = pivot + 1;
 % STEP 2 Finner indeks for inngående variabel
 [minxi i]= min(xstjerneB);
% STEP 3 Regner delta zN
Bindex=B_index';ii=Bindex(i);
deltazN = -(B\N)'*A(1:m,ii);
% STEP 4
for r = 1:n
  T = deltazN(r)./zstjerneN(r);
 s(r) = T;
end
[s j] = max(s);s = inv(s);
% STEP 5 Finner indeksen j til utgående variabel
j = N_index(j);
% STEP 6
deltaxB = B\N(1:m, j);
% STEP 7
t = xstjerneB(i)/deltaxB(i);
% STEP 8
xstjerneB = xstjerneB - t*deltaxB;zstjerneN = zstjerneN - s*deltazN;
Bindex=B_index';iny=Bindex(i);
% STEP 9
for k =1:m % bytter ut  utg. med inng. og danner ny basisindekser
    if B_index(k)==iny
        B_index(k)=j;
    end
end
```

```
for r =1:n % danner ny ikkebasisindekser
    if N_index(r)==j
        N_index(r)=iny;
    end
end
N=A(:,N_index); B=A(:,B_index);
% 2. Ny basis og ikke basis dual variabler
B_index=B_index';xstjerneBny=[B_index xstjerneB];% danner en ny hjelpematrise
%for å holde orden på indekser
for h=1:m
  if  xstjerneBny(h,1)==j % x(j) får verdien t
xstjerneBny(h,2)=t;
  end
end
xstjerneB =xstjerneBny(:,2);% ny x* er produsert
N_index=N_index';
zstjerneNny=[N_index zstjerneN];% danner en ny hjelpematrise
%for å holde orden på ikkebasisindekser
for h=1:n
  if  zstjerneNny(h,1)==iny
    zstjerneNny(h,2)=s;% z(i)=s
  end
end
zstjerneN =zstjerneNny(:,2);% lages ny z*
x = xstjerneB;z = zstjerneN;
% tester om løsningen optimal
[test a]=min(xstjerneB);
if xstjerneB >= 0
    antall_trinn = pivot
  optimal_solut = zeros(n,1);optimal_solut(B_index,:) = x;
 optimal_solut=optimal_solut(1:n);x=optimal_solut
 cN = cN(1:n);optimal_verdi = cN'*x
  return
end
cN = cN(1:n);B_index=B_index';N_index=N_index';
end
```

F Dual To-Fase Simpleksalgoritme.

Program 6 Dual To-Fase Simpleksalgoritme.

```
% Kjører DUAL ToFase SIMPLEKSALGORITME
 % STEP 1 Hvis XstjerneB < 0 så løsningen ikke optimal
iter = 0;
while 1==1
   iter = iter + 1;
 % STEP 2 Finner indeks for inngående variabel
 [minxi i]= min(xstjerneB);
% STEP 3 Regner delta zN
Bindex=B_index';ii=Bindex(i);deltazN =-(B\N)'*A(1:m,ii);
% STEP 4
for r = 1:n
  T = deltazN(r)./zstjerneN(r);
 s(r) = T;
end
[s j] = max(s);s = inv(s);
% STEP 5 Finner indeksen j til utgående variabel
j = N_index(j);
% STEP 6
deltaxB = B\N(1:m, j);
% STEP 7
t = xstjerneB(i)/deltaxB(i);
% STEP 8
xstjerne(j)=t;xstjerneB = xstjerneB - t*deltaxB;
zstjerne(j) = s;zstjerne = zstjerne(:);zstjerneN = zstjerneN - s*deltazN;
Bindex=B_index';iny=Bindex(i);
% STEP 9
for k =1:m % bytter ut indekser, utg. med inng. og danner ny basisindekser
    if B_index(k)==iny
        B_index(k)=j;
    end
end
for r =1:n % danner ny ikkebasisindekser
    if N_index(r)==j
        N_index(r)=iny;
    end
end
N=A(:,N_index); B=A(:,B_index);
% 2. Ny basis og ikke basis dual variabler
B_index=B_index';xstjerneBny=[B_index xstjerneB];% danner en ny hjelpematrise
%for å holde orden på indekser
```

```
for h=1:m
  if  xstjerneBny(h,1)==j % x(j) får verdien t
xstjerneBny(h,2)=t;
  end
end
xstjerneB =xstjerneBny(:,2);% ny x* er produsert
N_index=N_index';zstjerneNny=[N_index zstjerneN];% danner en
%ny hjelpematrise for å holde orden på ikkebasisindekser
for h=1:n
  if  zstjerneNny(h,1)==iny
    zstjerneNny(h,2)=s;% z(i)=s
  end
end
zstjerneN =zstjerneNny(:,2);% lages ny z*
x = xstjerneB;z = zstjerneN;
% tester om løsningen optimal
[test a]=min(xstjerneB);
if xstjerneB >= 0
  optimal_solut = zeros(n,1);optimal_solut(B_index,:) = x;
 optimal_solut=optimal_solut(1:n);x=optimal_solut;
 cN = cN(1:n);cB = zeros(n,1); cB=cNg;
iter;
b = xstjerneB;
N=A(:,N_index); B=A(:,B_index);
B_index=B_index;N_index=N_index;
zstjerneN =(inv(B)*N)'*cB(1:m);
 Primal_Tofase_Simpleks
  return
 end;
  cN = cN(1:n);B_index=B_index';N_index=N_index';
end
```

G Primal To-Fase Simpleksalgoritme.

Program 7 Primal To-Fase Simpleksalgoritme.

```
% STEP 1 Hvis zstjerneN < 0 så løsningen ikke optimal
iter=0;
while 1==1
    %for i=1:4
    iter = iter + 1;
 % STEP 2 Finner indeks for inngående variabel
 [minzj j]= min(zstjerneN);
% STEP 3 Regner delta xB
%Nindex=N_index
j=N_index(j);deltaxB =B\N(1:m, j);
% STEP 4
for r = 1:m
  T = xstjerneB(r)\deltaxB(r);
 t(r) = T;
end
[t i] = max(t);t = inv(t);
% STEP 5 Finner indeksen i til utgående variabel
i = B_index(i);
% STEP 6
deltazN =-(B\N)'(1:n,i);
% STEP 7
s = deltazN(j)\zstjerneN(j);
% STEP 8
xstjerneB = xstjerneB - t*deltaxB;
zstjerneN = zstjerneN - s*deltazN;
% STEP 9
for k =1:m % bytter ut indekser, utg. med inng. og danner ny basisindekser
    if B_index(k)==i
        B_index(k)=j;
    end
```

```
end
for r =1:n % danner ny ikkebasisindekser
    if N_index(r)==j
        N_index(r)=i;
    end
end
N=A(:,N_index);B=A(:,B_index);
% 2. Ny basis og ikke basis dual variabler
xstjerneBny=[B_index xstjerneB];% danner en ny hjelpematrise
%for å holde orden på indekser
for h=1:m
  if  xstjerneBny(h,1)==j % x(j) får verdien t
xstjerneBny(h,2)=t;
  end
end
xstjerneB =xstjerneBny(:,2);% ny x* er produsert
zstjerneNny=[N_index zstjerneN];% danner en ny hjelpematrise
%for å holde orden på ikkebasisindekser
for h=1:n
  if  zstjerneNny(h,1)==i
    zstjerneNny(h,2)=s;% z(i)=s
  end
end
zstjerneN =zstjerneNny(:,2);% lages ny z*
x = xstjerneB;z = zstjerneN;
% tester om løsningen optimal
[test a]=min(zstjerneN);
if test >= 0
    antall_trinn = iter
  optimal_soluti = zeros(n,1);
  optimal_soluti(B_index,:)=x;
  optimal_soluti=optimal_soluti(1:n);
  x=optimal_soluti
  cN=cNg;
  optimal_verdi = cN'*x
  return
 end;
 iter=iter
B_index=B_index';N_index=N_index';
end
```

H Primal Simpleksalgoritme.

Program 8 Primal Simpleksalgoritme.

```
% STEP 1 Hvis zstjerneN < 0 så løsningen ikke optimal
iter=0;
while 1==1
    iter = iter + 1;
 % STEP 2 Finner indeks for inngående variabel
 [minzj j]= min(zstjerneN);
% STEP 3 Regner delta xB
j=N_index(j);deltaxB = B\N(1:m, j);
% STEP 4
for r = 1:m
  T = xstjerneB(r)\deltaxB(r);
 t(r) = T;
end
[t i] = max(t);t = inv(t);
% STEP 5 Finner indeksen i til utgående variabel
i = B_index(i);
% STEP 6
deltazN = -(B\N)'*A(1:m,i);
% STEP 7
s = deltazN(j)\zstjerneN(j);
% STEP 8
xstjerneB = xstjerneB - t*deltaxB;
zstjerneN = zstjerneN - s*deltazN;
% STEP 9
for k =1:m % bytter ut indekser, utg. med inng. og danner ny basisindekser
    if B_index(k)==i
        B_index(k)=j;
    end
end
for r =1:n % danner ny ikkebasisindekser
    if N_index(r)==j
        N_index(r)=i;
    end
```

```
end
N=A(:,N_index);B=A(:,B_index);
% 2. Ny basis og ikke basis dual variabler
B_index=B_index';N_index=N_index';
xstjerneBny=[B_index xstjerneB];% danner en ny hjelpematrise
%for å holde orden på indekser
for h=1:m
  if  xstjerneBny(h,1)==j % x(j) får verdien t
xstjerneBny(h,2)=t;
  end
end
xstjerneB =xstjerneBny(:,2);% ny x* er produsert
zstjerneNny=[N_index zstjerneN];% danner en ny hjelpematrise
%for å holde orden på ikkebasisindekser
for h=1:n
  if  zstjerneNny(h,1)==i
    zstjerneNny(h,2)=s;% z(i)=s
  end
end
zstjerneN =zstjerneNny(:,2);% lages ny z*
x = xstjerneB;z = zstjerneN;
% tester om løsningen optimal
[test a]=min(zstjerneN);
if test >= 0
    antall_trinn = iter
  optimal_solut = zeros(n,1);
  optimal_solut(B_index,:)=x;
  optimal_solut=optimal_solui(1:n);
  x=optimal_solut
  cN=cNg;
  optimal_verdi = cBg'*x
  return
 end;
 iter=iter
B_index=B_index';N_index=N_index';
end
```

I Cholesky faktorisering.

Program 9 Cholesky funksjonen.

```
  function R =Cholesky(F)
%funksjonen som lager høyere triangular matrise R
ny=length(F); R=zeros(ny,ny);
 for i = 1:ny
    s=R(1:i-1,i); R(i,i)=sqrt(F(i,i)-s'*s);
    R(i,i+1:ny)=(F(i,i+1:ny)-s'*R(1:i-1,i+1:ny))/R(i,i);
end
```

Program 10 Funksjonen Forwardsolve.m

```
 % funksjonen som løser lav triangular system
function yny = Forwardsolve(R,bny)
 yny=bny(:); ny=length(bny);
 if ny==1
     yny=inv(R')*bny;
 else
    for k =1:ny
    yny(k)=(yny(k)-R(1:k-1,k)'*yny(1:k-1))/R(k,k);
    end
 end
```

Program 11 Funksjonen Backsolve.m

```
 % funksjonen som løser høyre triangular system
function xny = Backsolve (R,yny)
 yny=yny(:);xny=yny(:);ny=length(yny);
 if ny==1
     xny=yny*inv(R);
 else
    for k =ny:-1:1
    xny(k)=(xny(k)-R(k,k+1:ny)*xny(k+1:ny))/R(k,k);
    end
 end
```

Program 12 LP generator.

```
m=10;n=10;% disse verdiene kan endres etter behov
maxint=5;minint=-1;
A = minint+ceil(maxint.*rand(m,n));b=ceil(maxint.*rand(m,1));
c=minint+ceil(maxint.*rand(n,1));
```

J PF-algoritmen i generell form.

Gitt LP-problem i generell form:

$$\begin{aligned} \text{maksimer} \quad & c^T x \\ \text{f.at} \quad & \\ & a \leq Ax \leq b \\ & l \leq x \leq u. \end{aligned}$$

Innfører slakkvariablene:

$$\begin{aligned} \text{maksimer} \quad & c^T x \\ \text{f.at} \quad & \\ & Ax + f = b \\ & -Ax + p = -a \\ & x + t = u \\ & -x + g = -l \\ & f, p, t, g \geq 0. \end{aligned}$$

Dualet gitt ved

$$\begin{aligned} \text{minimer} \quad & b^T v - a^T q + u^T s - l^T h \\ \text{f.at} \quad & \\ & A^T(v - q) - (h - s) = c \\ & v, q, s, h \geq 0, \end{aligned}$$

og korresponderende komplementærslakk gitt ved:

$$\begin{aligned} f_i v_i &= 0 \quad i = 1, 2, \ldots, m, \\ p_i g_i &= 0 \quad i = 1, 2, \ldots, m, \\ t_j s_j &= 0 \quad j = 1, 2, \ldots, n, \\ g_j h_j &= 0 \quad j = 1, 2, \ldots, n. \end{aligned}$$

Neste steget er å introdusere primal-dual central path med parameter μ. Vi definerer et punkt på central path som primal-dual tillatt og som tilfredstiller μ-komplementæritets kriterier:

$$\begin{aligned} Ax + f &= b \\ f + p &= b - a \\ x + t &= u \\ -x + g &= -l \end{aligned}$$

$$\begin{aligned} A^T y + s - h &= c \\ y + q - v &= 0 \end{aligned}$$

$$\begin{aligned} FVe &= \mu e \\ PQe &= \mu e \\ TSe &= \mu e \\ GHe &= \mu e, \end{aligned}$$

hvor vi har innført ny dual variabel $y = v - q$. Vi har et ikkelineært likningssystem med $5n + 5m$ likninger og $5n + 5m$ ukjente. Det har en unik løsning i indre primal-dual rom:

$$(x, f, p, t, g, y, v, q, s, h) : f, p, t, g, v, q, s, h \geq 0. \tag{J.1}$$

Når μ nærmer seg null, konvergerer central path mot optimal løsning. Anta at vi har valgt μ, la $(x, \ldots, h)$ være et startpunkt, og $(x+\Delta x, \ldots, h+\Delta h)$ et punkt på central path som tilsvarer μ. Vi får da følgende likningssystem:

$$\begin{aligned} A\Delta x + \delta f &= b - Ax - f &&=: \rho \\ \Delta f + \Delta p &= b - a - f - p &&=: \alpha \\ \Delta x + \Delta t &= u - x - t &&=: \tau \\ -\Delta x + \Delta g &= -l + x - g &&=: \nu \end{aligned}$$

$$\begin{aligned} A^T\Delta y + \Delta s - \Delta h &= c - A^T y - s + h &&=: \sigma \\ \Delta y + \Delta q - \Delta v &= -y - q + v &&=: \beta \end{aligned}$$

$$\begin{aligned} FV^{-1}\Delta v + \Delta f &= \mu V^{-1}e - f - V^{-1}\Delta V\Delta f &&=: \gamma_f \\ QP^{-1}\Delta p + \Delta q &= \mu P^{-1}e - q - P^{-1}\Delta P\Delta q &&=: \gamma_q \\ ST^{-1}\Delta t + \Delta s &= \mu T^{-1}e - s - T^{-1}\Delta T\Delta s &&=: \gamma_s \\ HG^{-1}\Delta g + \Delta h &= \mu G^{-1}e - h - G^{-1}\Delta G\Delta h &&=: \gamma_h. \end{aligned}$$

Eliminerer Δv, Δp, Δg og Δt fra dette systemet og får et nytt på matriseform:

$$\begin{bmatrix} -TS^{-1} & & & & I & \\ & GH^{-1} & & & -I & \\ & & -PQ^{-1} & & I & \\ & & & & A & I \\ I & -I & & A^T & & \\ & & I & I & & VF^{-1} \end{bmatrix} \begin{bmatrix} \Delta s \\ \Delta h \\ \Delta q \\ \Delta y \\ \Delta x \\ \Delta f \end{bmatrix} = \begin{bmatrix} \hat{\tau} \\ \hat{\nu} \\ \hat{\alpha} \\ \rho \\ \sigma \\ \hat{\beta} \end{bmatrix}.$$

Eliminerer nå Δs, Δh og Δq fra det nye systemet:

$$\begin{bmatrix} & A & I \\ A^T & D & \\ I & & E \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \\ \Delta f \end{bmatrix} = \begin{bmatrix} \rho \\ \sigma + ST^{-1}\hat{\tau} - HG^{-1}\hat{\nu} \\ \hat{\beta} + QP^{-1}\hat{\alpha} \end{bmatrix},$$

hvor $D = ST^{-1} + HG^{-1}$ og $E = VF^{-1} + QP^{-1}$. Til slutt bruker vi pivotelementet E for å løse Δf:

$$\Delta f = E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha} - \Delta y),$$

og får redusert KKT system:

$$\begin{bmatrix} -E^{-1} & A \\ A^T & D \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} \rho - E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha} \\ \sigma + ST^{-1}\hat{\tau} - HG^{-1}\hat{\nu} \end{bmatrix}.$$

Algorithm 5 PF-algoritme i generell form

Bestem $(x, f, p, t, g, y, v, q, s, h)$ slik at $f, p, t, g, v, q, s, h > 0$

while ikke optimal **do**

$\rho = b - Ax - f$

$\sigma = c - A^T y - s + h$

$\gamma = fTv + p^T q + t^T s + g^T h$

$\mu = \delta \frac{\gamma}{n+m}$

$\gamma_f = \mu V^{-1} e - f$

$\gamma_q = \mu P^{-1} e - q$

$\gamma_s = \mu T^{-1} e - s$

$\gamma_h = \mu G^{-1} e - h$

$\hat{\tau} = u - x - t - TS^{-1}\gamma_s$

$\hat{\nu} = -l + x - g - GH^{-1}\gamma_h$

$\hat{\alpha} = b - a - f - p - PQ^{-1}\gamma_q$

$\hat{\beta} = -y - q + v + VF^{-1}\gamma_f$

$D = ST^{-1} + HG^{-1}$

$E = VF^{-1} + QP^{-1}$

Løs

$$\begin{bmatrix} -E^{-1} & A \\ A^T & D \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} \rho - E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha} \\ \sigma + ST^{-1}\hat{\tau} - HG^{-1}\hat{\nu} \end{bmatrix}$$

$\Delta f = E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha} - \Delta y)$

$\Delta s = -ST^{-1}(\hat{\tau} - \Delta x)$

$\Delta h = -HG^{-1}(\hat{\nu} + \Delta x)$

$\Delta q = -QP^{-1}(\hat{\alpha} - \Delta f)$

$\Delta v = VF^{-1}(\gamma_f - \Delta f)$

$\Delta p = PQ^{-1}(\gamma_g - \Delta q)$

$\Delta g = GH^{-1}(\gamma_h - \Delta h)$

$\Delta t = TS^{-1}(\gamma_s - \Delta s)$

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta f_j}{f_j}, -\frac{\Delta p_i}{p_i}, -\frac{\Delta t_i}{t_i}, -\frac{\Delta g_j}{g_j}, -\frac{\Delta v_j}{v_j}, -\frac{\Delta q_i}{q_i}, -\frac{\Delta s_i}{s_i}, -\frac{\Delta h_j}{h_j} \right\} \right)^{-1} \wedge 1$$

$$\begin{array}{llll} x \leftarrow x + \theta\Delta x, & y \leftarrow y + \theta\Delta y, & f \leftarrow f + \theta\Delta f, & v \leftarrow v + \theta\Delta v \\ p \leftarrow p + \theta\Delta p, & q \leftarrow q + \theta\Delta q, & t \leftarrow t + \theta\Delta t, & s \leftarrow s + \theta\Delta s \\ g \leftarrow g + \theta\Delta g, & h \leftarrow h + \theta\Delta h & & \end{array}$$

end while

Referanser

[Dahl08] G. Dahl. *A mini-introduction to convexity.* 2008.

[Bixb] R.E.Bixby. *Solving real-word linear programs: a decade and more of progress.* url= `http://www.ifi.uio.no/it/latex-links/BibTeX-lokal-guide.pdf` (2007-02-06).

[Vand08] R. Vanderbei. *Linear programming: foundations and extensions.* Springer, Third edition, 2008.

[BDat03] S.Boyd and J.Dattorro. *Alternating Projections* Stanford University, 2003.

[BoVa] S.Boyd and L.Vandenberghe *Convex Optimization* Cambridge University Press, Cambridge, 2004.

[Dahl04] G. Dahl. *An introduction to convexity, Kompendium* UiO, 2004. url=`http://folk.uio.no/geird`,

[Dant97] G.B. Dantzig and M. N. Thapa. *Linear Programming: Theory and Extensions* Springer, 1997.

[Karl91] H. Karloff. *Linear Programming* Birkhäuser Boston, 1991.

[Rene01] J. Renegar. *A Mathematical View of Interior-Point Methods in Convex Optimization* MPS-SIAM Series on Optimization, 2001.

[Wrig97] S. Wright. *Primal-Dual Interior-Point Methods* SIAM, 1997.

[Roo04] C. Roos, T. Terlaky and J.-P. Vial. *Interior Point Methods for Linear Optimization* Springer, Second Edition, 2006.

[Mars90] R. Marsten, R. Subramanian, M. Saltzman, I. Lustig and D. Shanno. *Interior Point Methods for Linear Programming: Just Call Newton, Lagrange, and Fiacco and McCormick!* The Institute of Management Sciences, 1990, 105-106.

[Lind06] T.Lindstrøm *Kalkulus* Universitetsforlaget, 3.utgave, 2006.

[ChGold59] W.Cheney and A.Goldstein *Proximity maps for convex sets.* Proceedings of the AMS, 10:448-450, 1959.

[Glu94] W.K.Glunt *An alternating projections method for certain linear problems in a Hilbert space* IMA Journal of Numerical Analysis (1995) 15, 291-305.

[Gou02] N.I.M. Gould *How good are projection methods for convex feasibility problems?* Oxford University, 2002

Figurer

Algoritmer

Programmer

Tabeller