Optimering og lineær programmering

Simplex- og ellipsoidealgoritmen

PROJEKTRAPPORT P2
GRUPPE A. 401

Aalborg Universitet
Føreste Studieår v/ Det Tekniske-Naturvidenskabelige Fakultet
Strandvejen 12-14 • 9000 Aalborg

Basis
Matematik
Strandvejen 12-14
9000 Aalborg
http://math.aau.dk

Titel:

Optimering og lineær programmering

Projekt:

P2-projekt

Projektperiode:

01/02-2019 - 27/05-2019

Projektgruppe:

A. 401

Deltagere:

Bjarke Brorsen Hebsgaard
Christian Boslev Søndergaard
Frida Holm
Ida Emilie Thomasine Stig Meyer
Rasmus Bod Olesen
Rasmus Skovborg Jensen

Vejleder:

Anne Marie Svane
Henrik Worm Routhe

Sidetal: 50 + Formalia og appendiks.
Afsluttet: 27/05-2019

Synopsis:

I løbet af 1940'erne begyndte en interesse for løsningen af en række optimeringsproblemer, der var begrænset af lineære uligheder, at bygge sig op. Disse problemer blev kaldt lineære programmeringsproblemer, og de beskrives, samt undersøges, i dette projekt. Der undesøges to algoritmer til løsningen af lineære programeringsproblemer: simplexalgoritmen og ellipsoidealgoritmen. Gennem dette projekt beskrives disse to algoritmer og sammenlignes med hinanden, derudover er simplexalgoritmen programmeret i Python3. Rapporten undersøger to tilgange til lineær optimering, som relaterer sig til hver af de to algoritmer, der beskrives i rapporten. Ved beskrivelsen af simplexalgoritmen anvendes en algebraisk tilgang til disse lineære programmeringsproblemer, mens ellipsoidealgoritmen håndterer problemerne ud fra en geometrisk synsvinkel.

Forord

Vi er en gruppe på seks matematikstuderende på andet semester, som har udarbejdet et P2 projekt omhandlende optimering og lineær programmering. Projektet er blevet udarbejdet i perioden 01/02-19 til 27/05-19. Vi har haft to vejledere, en til de matematiske problemstillinger og en bivejleder til problembaseret læring, hvor problemformulering og anvendelse af matematikken indgår. Vi vil gerne takke dem for at fremhæve fejl og mangler, så vi kunne forbedre os. Desuden var de altid behjælpelige og inspirerende, hvilket hjalp os med at komme på gode idéer til, hvordan projektet kunne komme i en ny retning. Det har være interessant at udarbejde og indsamle viden til projektet. God læselyst!

Underskrifter

Bjarke Brorsen Hebsgaard

Christian Boslev Søndergaard

Frida Holm

Ida Emilie Thomasine Stig Meyer

Rasmus Bod Olesen

Rasmus Skovborg Jensen

Indholdsfortegnelse

1 | Indledning

Sovietunionen og USA stod med en række af problemer i løbet af anden verdenskrig, der omhandlede logistikken af krigsførsel. Den sovjetiske økonom Leonid Kantorovich formulerede disse problemer som lineær programmeringsproblemer. Samtidigt i USA beskrev Tjalling Koopmans de samme problemer, som Frank Lauren Hitchcock lavede en metode til at løse. I 1975 modtog Kantorovich og Koopmans Nobelprisen i økonomi for deres introduktion af lineær programmering. Hitchcock modtog den ikke grundet hans død. [The Royal Swedish Academy of Sciences, 2019]. Det var dog først efter anden verdenskrig at lineær programmering fik sit navn. George Bernard Dantzig brugte lineær programmeringsproblemer til at lave programmer til det amerikanske luftvåben. Dette var før offentligheden, eller for den sags skyld Dantzig, kendte til computere, så når der diskuteres programmering, så har det intet med computerprogrammer at gøre, men derimod programmer i hæren, hvor optimering af hærens logistik har altafgørende betydning. Lineær programmering handler om at, ud fra lineære begrænsninger at få en optimal værdi givet en lineær funktion. Et eksempel på et lineært programmeringsproblem kunne være i landbruget omkring, hvad der bedst kan betale sig at plante på en mark, eller hvilke dyr der giver bedst udbytte.

Til løsningen af disse lineære programmeringsproblemer udviklede Dantzig simplexalgoritmen, som stadig er en af de centrale algoritmer brugt i dag. Algoritmen isolerer variable i problemets betingelser, og sætter dem ind i ligningen som det ønskes at optimere. Ved at vælge variable udvalgt af en pivotregel, forøges en konstant i problemets funktion. Der var dog et stort problem med algoritmen, nemlig at den for nogle problemer ville bruge mange beregninger. Antallet af beregninger som en algoritme maksimalt skal bruge for at bestemme en løsning kaldes dens kompleksiaatet, og den afhænger af størrelsen af inputtet. Programmet kørte hurtigt i praksis, men teoretisk set havde den en eksponentiel kompleksitet, $O(2^n)$. Der blev ikke lavet nogle bedre algoritmer indtil 1979, hvor Leonid Khachiyan udviklede ellipsoidealgoritmen. Simplexalgoritmen havde dog generalt kortere gennemløbstid, da store dele af algoritmen springes over for de fleste problemer. Ellisoidealgoritmen skal på den anden side altid igennem de fleste beregninger, og har derfor generelt længere gennemløbstid. Ellipsoidemetoden er derfor meget langsommere end simplexalgoritmen i praksis, men den har polynomiel kompleksitet, $O(n^4 \cdot log(\frac{1}{\varepsilon}))$, [Kalai and Kleitman, 1992]. Ellipsoidealgoritmen bestemmer den optimale løsning for et lineært programmeringsproblem, ved at danne ellipsoider over problemet og undersøge hvorvidt centrum af ellipsoiden er i problemet mulige mængde. Løsningsmetoderne danner motivationen bag dette projekt, og derfor vil disse algoritmer blive undersøgt. Ud fra dette kan et problem nu formuleres.

1.1 Problemformulering

Hvordan kan henholdsvis ellipsoide- og simplexalgoritmen anvendes til at løse lineære optimeringsproblemer, og er der et matematisk grundlag for at benytte den ene algoritme frem for den anden?

1.2 Afgrænsning

Det skal her nævnes at i 1984 blev en version af interior-point algoritmen udviklet, hvilket både er hurtig og har polynomiel kompleksitet, men denne vil ikke blive fokuseret på her.[Hurlbert, 2010]. I forhold til algoritmerne, kunne det være en mulighed at fokusere på geometrien bag simplexalgoritmen, men for at begrænse projektet, har vi valgt at undlade meget af geometrien bag denne algoritme, grundet tidsrammen og for at fokusere mere på algebraen bag simplex. For at beskrive disse algoritmer, så redegøres der for relevant lineær algebra og lineær programmering.

2 Elementær lineær algebra

I dette kapitel defineres basale matrixoperationer, samt anvendelsen af matricer til løsning af lineære ligningssystemer. Disse operationer anvendes gennem hele rapporten, og derfor er det vigtigt at fastlægge klare definitioner og termer. Dette kapitel er udarbejdet ved brug af [Geil, 2015], medmindre andet er angivet.

2.1 Matricer

En *matrix* består af m *rækker* og n *søjler* og kan sammenlignes lidt med en tabel, der kan udføres regneoperationer på. Definitionen på en matrix angives således:

Definition 2.1.1. *En matrix er en rektangulær liste af reelle tal. En matrix som har m rækker og n søjler, har dimensionen $m \times n$. I en matrix, A, kaldes den i'te række $\overrightarrow{r}_i$ og den j'te søjle kaldes $\overrightarrow{s}_j$.*

En matrix skrives på den generelle form:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}. \tag{2.1}$$

Bemærk her, hvordan $a_{i,j}$ betegner elementet, der findes i den i'te række og j'te søjle, og kaldes en *indgang* i matricen. Hvis $n = m$ siges matricen at være *kvadratisk*, hvis $n = 1$ siges matricen at være en *søjlevektor*, og hvis $m = 1$ siges matricen at være en *rækkevektor*.

Eksempel 2.1.2. En matrix med 3 rækker og 4 søjler er angivet:

$$A = \begin{bmatrix} 5 & -3 & 1 & 2 \\ \frac{5}{2} & -\frac{\pi}{7} & \frac{1}{3} & 2^2 \\ \pi & -9 & 40 & 0 \end{bmatrix}. \tag{2.2}$$

Indgangen $a_{3,1}$ kan således bestemmes ved at udvælge den tredje række og den første søjle. Derved fås at $a_{3,1} = \pi$.

Ved at bytte om på rækker og søjler ændres dimensionerne af en matrix således at en $m \times n$ matrix ændres til en $n \times m$ matrix. Denne operation kaldes en *transponering*. En matrix A, der bliver transponeret, skrives som A^T hvor der for enhver indgang i A^T gælder at $a^T_{i,j} = a_{j,i}$. Hvis $A = A^T$, siges A at være symmetrisk. og hvis $A = -A^T$, siges A at være skævsymmetrisk. Matrixprodukt af to transponerede matricer opfylder følgende $A^T B^T = (BA)^T$.

Hvis kun en del af en matrix skal undersøges, kan en såkaldt *undermatrix* dannes. En undermatrix af en matrix, A, dannes ved at fjerne en mængde af rækker og søjler. En undermatrix, hvor der er fjernet én række og én søjle, benævnes $A_{i,j}$, hvor i og j henholdsvis betegner den række og søjle, der er fjernet.

En *ledende indgang* i en matrix er den første indgang forskellig fra 0 i rækken, og en række kaldes en *nulrække*, hvis alle indgange i denne række er 0.

2.1.1 Matrixaddition og vektorer

Hvis to matricer har samme dimensioner kan de adderes. Dette gøres ved at lægge tilsvarende indgange sammen.

Definition 2.1.3. *Givet to $m \times n$ matricer A og B, er $A + B = C$, hvor $c_{i,j} = a_{i,j} + b_{i,j}$ for alle $i \in 1, \dots, m$ og $j \in 1, \dots, n$.*

Produktet af en matrix og en skalar er ligesom summen af matricer forholdsvis simpelt at definere. En matrix $C = k \cdot A$ hvor $k \in \mathbb{R}$ er defineret på følgende måde: $c_{i,j} = a_{i,j} \cdot k$ for alle $i = 1, \dots, m$ og $j = 1, \dots, n$.

Eksempel 2.1.4. To $m \times n$ matricer, A og B er givet ved

$$A = \begin{bmatrix} 5 & -3 & 1 & 2 \\ \frac{5}{2} & -\frac{\pi}{7} & \frac{1}{3} & 2 \\ \pi & -9 & 2 & 0 \end{bmatrix}, \tag{2.3}$$

$$B = \begin{bmatrix} 1 & -7 & 4 & \frac{1}{2} \\ -1 & \frac{4}{7} & \frac{2}{10} & 4 \\ \pi & 6 & 8 & 1 \end{bmatrix}. \tag{2.4}$$

Matricen $C = 2A + B$ bestemmes ud fra de operationer, der er fastlagt i dette kapitel. Derved bestemmes C til at være følgende

$$C = 2 \cdot \begin{bmatrix} 5 & -3 & 1 & 2 \\ \frac{5}{2} & 7 & \frac{1}{3} & 2 \\ \pi & -9 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 1 & -7 & 4 & \frac{1}{2} \\ -1 & -4 & \frac{2}{3} & 4 \\ \pi & 6 & 8 & 1 \end{bmatrix} = \begin{bmatrix} 11 & -13 & 6 & \frac{9}{2} \\ 4 & 10 & \frac{4}{3} & 8 \\ 3\pi & -12 & 12 & 1 \end{bmatrix}.$$

Produktet mellem en matrix og en vektor anvendes i mange tilfælde og derfor defineres denne regneoperation.

Definition 2.1.5. *Givet en $m \times n$ matrix A og en søjlevektor $\overrightarrow{v} = \left[v_1, v_2, \ldots, v_n\right]^T$, er matrix-vektor produktet af A og $\overrightarrow{v}$, skrevet som $A\overrightarrow{v}$, givet ved*

$$A\overrightarrow{v} = v_1\overrightarrow{a}_1 + v_2\overrightarrow{a}_2 + \cdots + v_n\overrightarrow{a}_n, \tag{2.5}$$

hvor a_n er den n'te søjlevektor.

På samme måde kan produktet mellem to matricer, $A_{m\times n}$ og $B_{n\times p}$, defineres til matrix-vektor produktet af A og samtlige søjlevektorer i B. Det vil sige at matrix-matrix produktet er givet som følgende $AB = [A\overrightarrow{b_1}, A\overrightarrow{b_2}, ..., A\overrightarrow{b_p}]$.

Ydermere benyttes notationen:

$$\overrightarrow{v} \geq \overrightarrow{w} \Leftrightarrow \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_i \end{bmatrix} \geq \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_i \end{bmatrix} \Leftrightarrow \begin{matrix} v_1 \geq w_1 \\ v_2 \geq w_2 \\ \vdots \\ v_i \geq w_i \end{matrix}.$$

Dette benyttes i projektet, da det er meget brugbart at kunne sammenligne vektorer. Hvis en vektor $\overrightarrow{v}$ sammenlignes med $\overrightarrow{0}$, hvilket er en vektor hvis koordinater alle er 0, antages $\overrightarrow{0}$ at have samme antal koordinater som $\overrightarrow{v}$. Længden af en vektor $\overrightarrow{a}$ skrives som $|\overrightarrow{a}| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$. Desuden er det gældende at $\overrightarrow{a} \cdot \overrightarrow{a} = |\overrightarrow{a}|^2$.

En kvadratisk matrix, hvis indgange i diagonalen $a_{i,i}$ er 1 og alle andre indgange 0, kaldes for en *identitetsmatrix* og benævnes I_n.

$$I_n = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}. \tag{2.6}$$

Identitetsmatricer har den egenskab, at enhver vektor, som kan ganges på identitetsmatricen vil give vektoren. En ortogonal matrix er også vigtig at definere, da ortogonale matricer har egenskaber, der er meget anvendelige for lineær programmering.

Definition 2.1.6. *En ortogonal matrix A opfylder, at $AA^T = I_n$*

Det er muligt at forene to eller flere matricer, horisontalt eller vertikalt, i en såkaldt *augmenteretmatrix.* Når matricer sættes sammen side om side, er det et krav, at de har samme antal rækker. Hvis matricerne sammensættes bund mod top er det et krav at matricerne har samme antal søjler. Såfremt en matrix er augmenteret, vil det være indikeret med en sort linje, der adskiller de oprindelige matricer, som illustreret her.

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = \left[\begin{array}{ccc|cc} a_{1,1} & a_{1,2} & a_{1,3} & b_{1,1} & b_{1,2} \\ a_{2,1} & a_{2,2} & a_{2,3} & b_{2,1} & b_{2,2} \\ \hline c_{1,1} & c_{1,2} & c_{1,3} & d_{1,1} & d_{1,2} \end{array}\right].$$

En *symmetrisk* matrix er en $n \times n$ matrix M, hvor $M^T = M$. En matrix M siges at være *positiv definit*, hvis M er en symmetrisk matrix, hvor $\overrightarrow{z}^T M \overrightarrow{z} > 0$ for alle $\overrightarrow{z} \neq \overrightarrow{0}$. En $n \times n$ matrix med *lineært uafhængige* søjler R kan bruges til at beskrive en positiv definit matrix M, da følgende gælder: $M = R^T R$. For at den skal være lineært uafhængig, så skal ligningen $R \cdot \overrightarrow{x} = \overrightarrow{0}$ kun have løsningen $\overrightarrow{x} = \overrightarrow{0}$.

2.1.2 Betingelser

Nogle af disse regneregler og definitioner er betinget af, at dimensionerne på matricerne er passende til de operationer, der udføres. Dette betyder, at addition af to matricer kun kan udføres, når matricerne har samme antal søjler og samme antal rækker. Det betyder også, at i tilfældet af matrix-vektor produkt, at antallet af koordinater i vektoren, er lig med antallet af søjler i matricen.

2.2 Lineære ligningssystemer

Lineære ligningssystemer indgår ofte i komplekse optimeringsproblemer med mange forskellige variable. Det er meget nyttigt at kunne bestemme løsninger til sådanne ligningssystemer, da disse problemer ofte optræder i erhvervslivet.

Definition 2.2.1. ***Lineær ligning***
En lineær ligning med n variable er en ligning på formen $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$, hvor $x_1, x_2, \ldots, x_n$ er variable, og $a_1, a_2, \ldots, a_n$ er koefficienter i $\mathbb{R}$. Desuden er b en konstant i $\mathbb{R}$ og kaldes for det konstante led.

Lineære ligningssystemer er et sæt af m lineære ligninger med n fælles variable. Koefficienterne kan være lig 0, og dette medfører, at udvalgte variable ikke påvirker ligningen.
Et generelt lineært ligningssystem kan skrives som:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1. \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2. \\ &\vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n &= b_m. \end{aligned} \tag{2.7}$$

Til at repræsentere et lineært ligningssystem, kan en *koefficientmatrix* anvendes, hvilket er en matrix med alle koefficienterne fra ligningssystemet, hvor hver række i matricen indeholder koefficienterne fra samme ligning. Dermed tilhører koefficienterne i samme søjle den samme variabel. Matricens generelle form er vist via matricen A_k i Ligning (2.8). Den augmenteredematrix af A_k og $\overrightarrow{b}$, kaldes en *totalmatrix* og skrives $A_t = \left[A_k \mid \overrightarrow{b}\right]$. Den augmenterede matrix A_t i Ligning (2.9) beskriver den generelle form for en totalmatrix.

$$A_k = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}. \quad (2.8) \qquad A_t = \left[\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & b_m \end{array}\right]. \quad (2.9)$$

Det generelle Ligningssystem (2.7) er ækvivalent med ligningen $A_k \cdot \overrightarrow{x} = \overrightarrow{b}$, hvor ligningssystemet er vist på totalmatrixform i Ligning (2.9). Her er tallene i den første søjle de koefficienter, som svarer til variablen x_1, den anden søjle svarer til x_2, indtil den andensidste søjle, der svarer til x_n. Den sidste søjle indeholder de konstante led.

2.2.1 Matrix rækkeoperationer

Om matricer, der repræsenterer lineære ligningssystemer, vil følgende operationer ikke ændre løsningsmængden:

- Ombytning: En række kan bytte plads med en anden række.
- Skalarmultiplikation: En række kan ganges igennem med en skalar forskellig fra 0.
- Udskiftning: En række gange en skalar kan lægges til en anden række.

Disse regneregler er gyldige, da de samme regneregler gælder for lineære ligningssystemer, hvor det er tilladt at ændre på rækkefølgen af ligningerne. Det, at gange en række igennem med en skalar, svarer til at gange alle led i en lineær ligning med samme tal. Når en række lægges til en anden række, vil det tilsvarende i et lineært ligningssystem være at lægge samme tal til på begge sider. Det er vigtigt at notere, at alle regnereglerne er reversible.

2.2.2 Trappeform og Gauss-elimination

For at bestemme løsningen til et lineært ligningssystem omdannes lingingssystemets matrix til *trappeform*. I denne sektion gennemgås, hvad der menes med trappeform, *reduceret trappeform* og algoritmen *Gauss-elimination*, der omdanner en hvilken som helst matrix til en matrix på reduceret trappeform.

Definition 2.2.2. *En matrix er på trappeform hvis og kun hvis matricen opfylder følgende:*

1. *Alle ikke-nulrækker ligger over alle nulrækker.*
2. *Den ledende indgang i alle ikke-nulrækker ligger i en søjle til højre for søjlen med den ledende indgang, for alle rækker over. Dette medfører, at alle indgange under en ledende indgang er 0.*

En matrix er på reduceret trappeform hvis og kun hvis de tidligere betingelser, samt følgende betingelser gælder.

3. *Hvis en søjle indeholder den ledende indgang af en række, så er alle andre indgange i søjlen 0.*
4. *Den ledende indgang af hver ikke-nulrække er 1.*

I en matrix på trappeform, kaldes den ledende indgang i hver række en *pivotindgang*, og søjlen, som pivotindgangen ligger i, kaldes en *pivotsøjle*, og tilsvarende er rækken med pivotindgangen en *pivotrække*. En variabel, som tilhører en pivotsøjle, kaldes en *bunden variabel*, og en variabel, som ikke hører til en pivotsøjle, kaldes for en *fri variabel*.
Først bringes totalmatricen på trappeform ved hjælp af rækkeoperationer. Herved ændres løsningsmængden ikke. For en totalmatrix på trappeform kan antallet af løsninger bestemmes.

- Ligningssystemet har én løsning, hvis der er pivot i alle søjler, undtagen den sidste.
- Hvis ligningssystemet har pivot i sidste søjle, findes der ingen løsninger.
- Hvis der ikke er pivot i to eller flere søjler, hvoraf den ene er den sidste, har ligningssystemet uendelig mange løsninger.

Er der pivot i alle søjler, bortset fra den sidste, når totalmatricen er på trappeform, regnes der videre til reduceret trappeform, som svarer til identitetsmatricen, I_n, tilføjet en ekstra søjle, $\overrightarrow{b}^*$ til sidst. Derfor giver det ligningen $I_n \cdot \overrightarrow{x} = \overrightarrow{b}^*$, hvilket betyder, at søjle $\overrightarrow{b}^*$ er løsningen til ligningssystemet.
Hvis der er pivot i sidste søjle, vil det betyde $0 \cdot x_1 + 0 \cdot x_2 + \cdots + 0 \cdot x_n = b_i^*$. Denne ligning har ingen løsninger, eftersom der ikke findes nogen variable, der opfylder ligningen, da $b_i^* \neq 0$.
Når der er mindst en ikke-pivotsøjle udover den sidste søjle, opstår der uendelig mange løsninger, da de variable, som ikke-pivotsøjlerne repræsenterer, vil være frie variable. Dette kan ses i den reducerede trappeform, hvor der ikke vil være pivotindgange i alle søjler. Da der er frie variable, så vil der være løsninger svarende til alle værdier, som de frie variabler kan tage, hvilket er uendeligt mange.

For at omdanne en totalmatrix til reduceret trappeform ved hjælp af rækkeoperatiorne kan Gauss-elimination anvendes. Denne algoritme har 6 trin, som nu uddybes.

1. Find den første ikke-nulsøjle. Dette er en pivotsøjle. Vælg derefter pivotpositionen til at være den første indgang i pivotsøjlen.

2. Vælg en given ikke-nulindgang i pivotsøjlen, og brug ombytning til at flytte den hen til pivotpositionen, hvis nødvendigt.
3. Brug udskiftning til at ændre alle indgange under pivotpositionen i pivotsøjlen til 0.
4. Ignorér rækken med pivotpositionen og gentag trin 1-4 indtil matricen er på trappeform.
5. Nu skal skalarmultiplikation anvendes, så alle ledende indgange bliver 1.
6. Brug udskiftning til at ændre alle indgange over de ledende indgange til 0.

Eksempel 2.2.3. Nu opstilles et eksempel på et ligningssystem.

$$\begin{aligned} 4x_1 + 9x_2 + 2x_3 &= 7 \\ 2x_1 + 4x_2 + 8x_3 &= 4 \\ 8x_1 + 3x_2 + 5x_3 &= -3. \end{aligned} \tag{2.10}$$

Dette ligningssystem repræsenteres nu ved en totalmatrix.

$$A_t = \left[\begin{array}{ccc|c} 4 & 9 & 2 & 7 \\ 2 & 4 & 8 & 4 \\ 8 & 3 & 5 & -3 \end{array}\right]. \tag{2.11}$$

Nu anvendes Gauss-elimination på denne matrix. Da der ikke er en nulsøjle, så er $a_{1,1}$ pivotpositionen og trin 1 er færdiggjort. Desuden bliver intet gjort i trin 2, da ombytning ikke er nødvendigt. Udskiftning udføres to gange, da trin 3 anvender det én gang, og trin 4 påtvinger endnu en gennemgang af trin 1-3, hvor $a_{2,2}$ bliver pivotpositionen og endnu en udskiftning udføres igen.

$$\left[\begin{array}{ccc|c} 4 & 9 & 2 & 7 \\ 2 & 4 & 8 & 4 \\ 8 & 3 & 5 & -3 \end{array}\right] \xrightarrow[-2\cdot r_1 + r_3 \to r_3]{-\frac{1}{2}\cdot r_1 + r_2 \to r_2} \left[\begin{array}{ccc|c} 4 & 9 & 2 & 7 \\ 0 & -\frac{1}{2} & 7 & \frac{1}{2} \\ 0 & -15 & 1 & -17 \end{array}\right] \xrightarrow{-30\cdot r_2 + r_3 \to r_3} \left[\begin{array}{ccc|c} 4 & 9 & 2 & 7 \\ 0 & -\frac{1}{2} & 7 & \frac{1}{2} \\ 0 & 0 & -209 & -32 \end{array}\right]. \tag{2.12}$$

Nu anvendes skalarmultiplikation for at udføre trin 5. Her er skalaren, der ganges med i hver række, den inverse af rækkens ledende indgang.

$$\left[\begin{array}{ccc|c} 1 & \frac{9}{4} & \frac{1}{2} & \frac{7}{4} \\ 0 & 1 & -14 & -1 \\ 0 & 0 & 1 & \frac{32}{209} \end{array}\right]. \tag{2.13}$$

Til sidst udføres trin 6, så udskiftning er her nødvendig.

$$\left[\begin{array}{ccc|c} 1 & \frac{9}{4} & \frac{1}{2} & \frac{7}{4} \\ 0 & 1 & -14 & -1 \\ 0 & 0 & 1 & \frac{32}{209} \end{array}\right] \xrightarrow[-\frac{1}{2}\cdot r_3 + r_1 \to r_1]{14\cdot r_3 + r_2 \to r_2} \left[\begin{array}{ccc|c} 1 & \frac{9}{4} & 0 & \frac{1399}{836} \\ 0 & 1 & 0 & \frac{239}{209} \\ 0 & 0 & 1 & \frac{32}{209} \end{array}\right] \xrightarrow{-\frac{9}{4}\cdot r_2 + r_1 \to r_1} \left[\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{188}{209} \\ 0 & 1 & 0 & \frac{239}{209} \\ 0 & 0 & 1 & \frac{32}{209} \end{array}\right]. \tag{2.14}$$

Derved kan løsningen til ligningssystemet nu opskrives.

$$x_1 = -\frac{188}{209}, x_2 = \frac{239}{209}, x_3 = \frac{32}{209}. \tag{2.15}$$

2.2.3 Invertible matricer

En $n \times n$ matrix A kaldes *invertibel*, hvis der eksisterer en matrix B, sådan at $AB = BA = I_n$. Hvis sådan en matrix eksisterer så er den $A's$ inverse matrix, og kaldes $B = A^{-1}$. Hvis en matrix kan rækkereduceres til identitetsmaticen, så er det ensbetydende med at den er invertibel.

Eksempel 2.2.4. Lad A være matricen

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 6 & 7 \end{bmatrix}.$$

Nu skal A^{-1} bestemmes. Der dannes en augmenteret matrix $[A, I_3]$, som rækkereduceres til $[I_3, B]$. Her vil $B = A^{-1}$.

$$[A, I_3] = \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 & 1 & 0 \\ 1 & 6 & 7 & 0 & 0 & 1 \end{array}\right] \xrightarrow[r_3 - r_1 \to r_3]{r_2 - 3\cdot r_1 \to r_2} \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & -4 & -8 & -3 & 1 & 0 \\ 0 & 4 & 4 & -1 & 0 & 1 \end{array}\right]$$

$$-\tfrac{1}{4}r_2 \to r_2 \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 2 & \frac{3}{4} & -\frac{1}{4} & 0 \\ 0 & 4 & 4 & -1 & 0 & 1 \end{array}\right] \xrightarrow{r_3 - 4\cdot r_2 \to r_3} \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 2 & \frac{3}{4} & -\frac{1}{4} & 0 \\ 0 & 0 & -4 & -4 & 1 & 1 \end{array}\right]$$

$$-\tfrac{1}{4}r_3 \to r_3 \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 2 & \frac{3}{4} & -\frac{1}{4} & 0 \\ 0 & 0 & 1 & 1 & -\frac{1}{4} & -\frac{1}{4} \end{array}\right] \xrightarrow[r_1 - 3\cdot r_3 \to r_1]{r_2 - 2\cdot r_3 \to r_2} \left[\begin{array}{ccc|ccc} 1 & 2 & 0 & -2 & \frac{3}{4} & \frac{3}{4} \\ 0 & 1 & 0 & -\frac{5}{4} & \frac{1}{4} & \frac{1}{2} \\ 0 & 0 & 1 & 1 & -\frac{1}{4} & -\frac{1}{4} \end{array}\right]$$

$$\xrightarrow{r_1 - 2\cdot r_2 \to r_1} \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & -\frac{1}{4} \\ 0 & 1 & 0 & -\frac{5}{4} & \frac{1}{4} & \frac{1}{2} \\ 0 & 0 & 1 & 1 & -\frac{1}{4} & -\frac{1}{4} \end{array}\right]. \quad A^{-1} = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{5}{4} & \frac{1}{4} & \frac{1}{2} \\ 1 & -\frac{1}{4} & -\frac{1}{4} \end{bmatrix}.$$

Herfra er A^{-1} nu bestemt.

2.2.4 Pivotering

Pivoteres en indgang svarer det til at ændre indgangen til 1 og alle andre indgange i samme søjle til nul. De samme rækkeoperationer, som benyttes i Gauss-elimination, kan anvendes til dette formål. Først anvendes udskiftning sådan at søjlens andre indgange er nul, og derefter anvendes skalarmultiplikation til at fremtvinge et ettal i den pivoterede indgang.

Eksempel 2.2.5. Der ønskes nu at pivotere indgangen $a_{2,2}$ i matricen A.

$$A = \left[\begin{array}{ccc|c} 1 & 6 & 6 & 6 \\ 3 & 2 & 2 & 3 \\ 2 & 2 & 5 & 6 \end{array}\right] \xrightarrow[r_3 - r_2 \to r_2]{r_1 - 3 \cdot r_2 \to r_1} \left[\begin{array}{ccc|c} -8 & 0 & 0 & -3 \\ 3 & 2 & 2 & 3 \\ -1 & 0 & 3 & 3 \end{array}\right] \xrightarrow{\frac{1}{2} \cdot r_2 \to r_2} \left[\begin{array}{ccc|c} -8 & 0 & 0 & -3 \\ \frac{3}{2} & 1 & 1 & \frac{3}{2} \\ -1 & 0 & 3 & 3 \end{array}\right]. \tag{2.16}$$

Derved er $a_{2,2}$ pivoteret.

3 | Lineær programmering

Lineær programmering er hovedemnet i dette projekt og derfor defineres i dette kapitel, hvad der menes med et lineært programmeringsproblem. Dette kapitel er udarbejdet ved brug af [Bertsimas and Tsitsiklis, 1997] og [Hurlbert, 2010], medmindre andet er angivet.

3.1 Introduktion til Lineær Programmering

Det ønskes at maksimere $z = \overrightarrow{c} \cdot \overrightarrow{x} = \sum_{i=1}^{n} c_i \cdot x_i$, som kaldes *objektfunktionen*, hvor $\overrightarrow{c} = [c_1, \dots, c_n]$ er en *objektvektor* og $\overrightarrow{x} = [x_1, \dots, x_n]^T$. En vektor $\overrightarrow{x}$, kaldet en *løsning*, er begrænset af lineære ligheder og uligheder. Lad M_1, M_2, M_3 være endelige indeksmængder. For ethvert i i henholdsvis M_1, M_2, M_3 hører der en n-dimensionel rækkevektor $\overrightarrow{a}_i$ og en skalar $b_i \in \mathbb{R}$, der danner den i'te begrænsning. Lad $N_1, N_2 \subseteq \{m \in \mathbb{N} | m \leq n\}$ være indeksmængder der indikerer at $x_j \geq 0$ eller $x_j \leq 0$ for j i henholdsvis N_1 og N_2. Optimeringsproblemet lyder nu

$$
\begin{aligned}
\text{Maksimér: } & z = \overrightarrow{c} \cdot \overrightarrow{x} & & \\
\text{betinget af: } & \overrightarrow{a}_i \cdot \overrightarrow{x} \geq b_i, & & i \in M_1 \\
& \overrightarrow{a}_i \cdot \overrightarrow{x} \leq b_i, & & i \in M_2 \\
& \overrightarrow{a}_i \cdot \overrightarrow{x} = b_i, & & i \in M_3 \\
& x_j \leq 0, & & j \in N_1 \\
& x_j \geq 0, & & j \in N_2.
\end{aligned}
\tag{3.1}
$$

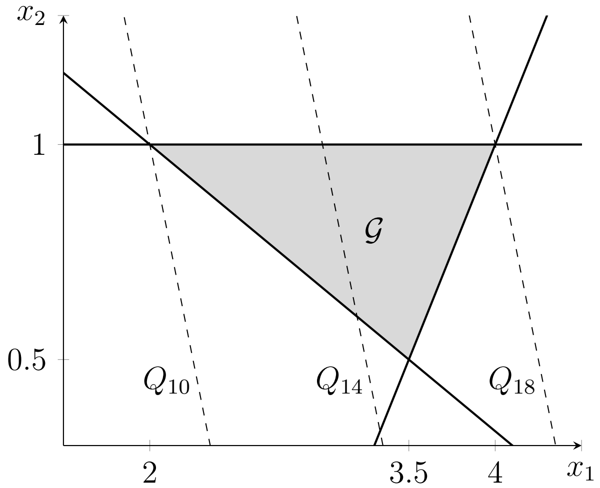
En vektor $\overrightarrow{x}$ er en *mulig løsning*, hvis den opfylder alle betingelser. Hvis en vektor $\overrightarrow{x}$ ikke opfylder alle betingelser kaldes den derfor en *ikke-mulig løsning*. Mængden af alle mulige løsninger kaldes for *den mulige mængde*. Hvis j hverken er i N_1 eller N_2 siges variablen x_j at være en *fri variabel*. Hvis et optimeringsproblem ikke har nogen mulig løsning, så kaldes det et *ikke-muligt problem*. Der er også tilfælde, hvor et problem har en ubegrænset mulig mængde, og hvor ingen af dem er en optimal løsning. Disse problemer kaldes *ubegrænsede problemer*.

En mulig løsning $\overrightarrow{x}'$ er en *optimal løsning* i et maksimeringsproblem, hvis den opfylder at $\overrightarrow{c} \cdot \overrightarrow{x}' \geq \overrightarrow{c} \cdot \overrightarrow{x}$ for alle $\overrightarrow{x}$ i den mulige mængde. I minimeringsproblemer er $\overrightarrow{c} \cdot \overrightarrow{x}' \leq \overrightarrow{c} \cdot \overrightarrow{x}$ for alle $\overrightarrow{x}$. Den optimale værdi vil derudfra være givet ved $z' = \overrightarrow{c} \cdot \overrightarrow{x}'$. Fremover betegnes alle optimale objektfunktioner og løsninger med en apostrof.

Eksempel 3.1.1. Her ønskes at løse optimeringsproblemet.

$$\begin{aligned} \text{Minimér: } z &= 4x_1 + 2x_2 \\ \text{betinget af: } \quad & x_1 - x_2 \leq 3 \\ & x_1 + 3x_2 \geq 5 \\ & x_2 \leq 1 \\ & x_1, \quad x_2 \geq 0. \end{aligned} \tag{3.2}$$

Disse begrænsninger kan indtegnes i et koordinatsystem, set på Figur 3.1. Først undersøges ekstrema i den mulige mængde $\mathcal{G}$. Idet z er lineær, vides det, at alle ekstrema i $\mathcal{G}$ vil ligge på randen af området. Nu undersøges forskellige konturlinjer for z. Linjerne er givet ved $Q_k : 4x_1 + 2x_2 = k$ for $k \in \mathbb{R}$. Nu indtegnes linjen Q_k for stigende værdier af k indtil linjen lige præcis rammer $\mathcal{G}$. Ved at lokalisere linjens skæringspunkt med $\mathcal{G}$ bestemmes så den optimale løsning til problemet, der i dette tilfælde vil være $x_1 = 2$ og $x_2 = 1$.



Figur 3.1: Illustration af uligheder med indtegnede konturlinjer for forskellige k. Det skal dog bemærkes at grafen ikke viser origo, men i stedet er koncentreret omkring den mulige mængde.

Det er muligt at illustrere et lineært optimeringsproblem ved et koordinatsystem, som i Eksempel 3.1.1, men hvis der er over 3 variable er det ikke muligt, da vi ikke kan fremstille en illustration på over 3 dimensioner. Der vil blive præsenteret optimeringsproblemer senere i projektet som har flere end 3 variable, hvor det ikke er muligt at illustrere dem, men de vil blive løst ved brug af andre metoder.

3.2 Standardform og dualitet

Der defineres, hvad et lineært programmeringsproblem på *standardform* er. Det er anvendeligt at kunne omforme ethvert problem til et problem på standardform, idet der senere i projektet vil blive formuleret en metode til at løse lineære programmeringsproblemer på standardform.

Definition 3.2.1. *Et problem er på standardform, hvis det er formuleret på følgende form:*

$$\begin{aligned} \textit{Maksimér: } z = \vec{c} \cdot \vec{x} & \\ \textit{betinget af:} \quad A \cdot \vec{x} &\leq \vec{b} \\ \vec{x} &\geq \vec{0}\,. \end{aligned} \tag{3.3}$$

Det er brugbart at kunne omformulere lineære programmeringsproblemer af typen beskrevet ved Ligningerne (3.1) til et problem på standardform. Dertil anvendes følgende operationer:

1. En lighedsbegrænsning $\vec{a}_i \cdot \vec{x} = b_i$ kan deles op i to ulighedsbegrænsninger $\vec{a}_i \cdot \vec{x} \leq b_i$ og $\vec{a}_i \cdot \vec{x} \geq b_i$.
2. En ulighedsbegrænsning af typen $\vec{a}_i \cdot \vec{x} \geq b_i$ kan skrives på formen $-\vec{a}_i \cdot \vec{x} \leq -b_i$.
3. En ulighed af typen $x_j \leq 0$ eller $x_j \geq 0$ er et specialtilfælde af uligheden $\vec{a}_i \cdot \vec{x} \leq b_i$ og $\vec{a}_i \cdot \vec{x} \geq b_i$, hvor $\vec{a}_i$ er den j'te enhedsvektor og $b_i = 0$.
4. Enhver fri variabel x_j kan skrives som $x_j = x_j^+ - x_j^-$ hvor $x_j^+, x_j^- \geq 0$.

Endvidere er det muligt at ændre ulighedsbegrænsninger til lighedsbegrænsninger ved brug af såkaldte *slackvariable*, som er nødvendige i forbindelse med løsningsalgoritmer. Lad en ulighedsbetingelse $\sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i$ være givet. Om slackvariablen s_i gælder der: $\sum_{j=1}^{n} a_{i,j} \cdot x_j + s_i = b_i$ og $s_i \geq 0$. Det vil sige, at følgende problem kan opstilles:

$$\begin{aligned} \text{Maksimér: } z = \vec{c} \cdot \vec{x} & \\ \text{betinget af:} \quad A \cdot \vec{x} + \vec{s} &= \vec{b} \\ \vec{x}, \vec{s} &\geq \vec{0}\,. \end{aligned} \tag{3.4}$$

Her er $\vec{s}$ en vektor der indeholder alle slackvariable. Hvis et problem er af typen beskrevet i Ligning (3.4), siges det at være på *slackform*. En løsning, som opfylder Ligningssystem (3.3), vil sammen med slackvariablene også opfylde Ligningssystem (3.4), desuden vil værdien af objektfuntkionen i begge problemer være den samme, når løsningen indsættes.

Eksempel 3.2.2. Der tages udgangspunkt i Eksempel 3.1.1, og det ønskes omformet til slackform. Problemet omformes først til standardform ved brug af operationerne fra tidligere i kapitlet.

$$\begin{aligned} \text{Maksimér: } & z = 4x_1 + 2x_2 \\ \text{betinget af: } & x_1 - x_2 + s_1 = 3 \\ & -x_1 - 3x_2 + s_2 = -5 \\ & x_2 + s_3 = 1 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0. \end{aligned} \tag{3.5}$$

Idet problemet nu stemmer overens med Ligning (3.4)

3.2.1 Svag dualitet

I dette afsnit er der, udover de to forudnævnte kilder, også blevet benyttet [Ekeocha et al., 2018]. Hvis et lineært programmeringsproblem er et minimeringsproblem, så er det ikke just nemt at ændre. Betragt de lineære programmeringsproblemer.

$$\begin{aligned} \text{Maksimér: } & z = 4x_1 + 3x_2 - x_3 \\ \text{betinget af: } & x_1 + 7x_3 \leq 5 \\ & x_2 + 5x_3 \leq 20 \\ & x_1 + x_3 \leq 4 \\ & x_1, x_2, x_3 \geq 0. \end{aligned} \tag{3.6}$$

$$\begin{aligned} \text{Minimér: } & w = 5y_1 + 20y_2 + 4y_3 \\ \text{betinget af: } & y_1 + y_3 \geq 4 \\ & y_2 \geq 3 \\ & 7y_1 + 5y_2 + y_3 \geq -1 \\ & y_1\ y_2\ y_3 \geq 0. \end{aligned} \tag{3.7}$$

Bemærk, at Problem (3.6) er et maksimeringsproblem med objektvektor $\overrightarrow{c}$, hvor begrænsningerne er af formen $A \cdot \overrightarrow{x} \leq \overrightarrow{b}$, hvorimod (3.7) er et minimeringsproblem med objektvektor $\overrightarrow{b}$, hvor begrænsningerne er af formen $A^T \cdot \overrightarrow{y} \geq \overrightarrow{c}^T$. Bemærk dog, at uligheden $\overrightarrow{x} \geq \overrightarrow{0}$ er uændret. Løsningen $\overrightarrow{x}$ er ændret til $\overrightarrow{y}$, hvilket blot er for at kende forskel på dem ($\overrightarrow{x}$ og $\overrightarrow{y}$ er mulige løsninger til programmeringsproblemet). Desuden er objektvektoren $\overrightarrow{c}$ blevet byttet ud med $\overrightarrow{b}$. Her siges (3.7) at være *dualen* til (3.6), som kaldes *primalen*. Primalens objektfunktion har n variable og m begrænsninger, hvor begrænsningerne sætter en øvre grænse for maksimeringen. Derimod har dualens objektfunktion m variable og n begrænsninger, som omvendt sætter en nedre grænse for minimeringen. Derved kan *primal-* og *dualformen* opstilles på generel form.

Primalform

$$\begin{aligned} \text{Maksimér: } & z = \overrightarrow{c} \cdot \overrightarrow{x} \\ \text{betinget af: } & A \cdot \overrightarrow{x} \leq \overrightarrow{b} \\ & \overrightarrow{x} \geq \overrightarrow{0}, \end{aligned} \tag{3.8}$$

Dualform

$$\begin{aligned} \text{Minimér: } & w = \overrightarrow{b}^T \cdot \overrightarrow{y} \\ \text{betinget af: } & A^T \cdot \overrightarrow{y} \geq \overrightarrow{c}^T \\ & \overrightarrow{y} \geq \overrightarrow{0}. \end{aligned} \tag{3.9}$$

Lad $\overrightarrow{x}$ og $\overrightarrow{y}$ være mulige løsninger til (3.8) og (3.9). Ud fra disse kan en ulighed nu opstilles.

$$\begin{aligned} z = \overrightarrow{c} \cdot \overrightarrow{x} \leq \overrightarrow{y}^T \cdot A \cdot \overrightarrow{x} \leq \overrightarrow{y}^T \cdot \overrightarrow{b} = w \\ \overrightarrow{x}, \overrightarrow{y} \geq \overrightarrow{0}. \end{aligned} \tag{3.10}$$

Derved er $z \leq w$ gældende for alle $\overrightarrow{x}$ og $\overrightarrow{y}$. Herudfra må det også være gældende, at den optimale værdi af primalens objektfunktion er mindre end dualens, altså $z' \leq w'$. Dette kaldes *svag dualitet*. *Stærk dualitet* viser, at $z' = w'$, men dette vil først blive bevist i Afsnit 4.4.

4 | Simplexalgoritmen

Simplexalgoritmen benyttes til at løse lineære programmeringsproblemer, hvor slackform er forudsat. Dette kapitel er udarbejdet ud fra [Hurlbert, 2010], medmindre andet er angivet. Der gennemgås et eksempel for at forstå idéen bag algoritmen, før de forskellige skridt formaliseres.

I denne sammenhæng er det vigtigt at definere, hvad der menes med en *basal variabel* og en *ikke-basal variabel*. Betragt det lineære programmeringsproblem på slackform.

$$\begin{aligned} &\text{Maksimér: } z = \overrightarrow{c} \cdot \overrightarrow{x} \\ &\text{betinget af: } \quad A \cdot \overrightarrow{x} + \overrightarrow{s} = \overrightarrow{b} \\ &\qquad\qquad\qquad \overrightarrow{x}, \overrightarrow{s} \geq \overrightarrow{0}\,. \end{aligned} \tag{4.1}$$

Dette kan omformes til følgende problem, der siges at være på *konkordansform*:

$$\begin{aligned} &\text{Maksimér: } z = v + \overrightarrow{c} \cdot \overrightarrow{x}_\rho \\ &\text{betinget af: } \quad \overrightarrow{x}_\beta = \overrightarrow{b} - A \cdot \overrightarrow{x}_\rho \\ &\qquad\qquad\qquad \overrightarrow{x}_\rho, \overrightarrow{x}_\beta \geq \overrightarrow{0}\,. \end{aligned} \tag{4.2}$$

Bemærk her at $\overrightarrow{x}$ erstattes med $\overrightarrow{x}_\rho$ og $\overrightarrow{s}$ erstattes med $\overrightarrow{x}_\beta$. Her siges variablene på venstresiden, undtagen z, at være basale variable og variablene på højresiden at være ikke-basale variable. Desuden kaldes v for den *basale objektfunktion*, og $\overrightarrow{x}_\rho$ er løsningen til konkordansen. Vektoren med de basale variable betegnes $\overrightarrow{x}_\beta$, hvor β er en indeksmængde der består af de basale variables indeks. Vektoren med de ikke-basale variable betegnes $\overrightarrow{x}_\rho$, hvor ρ indeholder alle ikke-basale variables indeks. Ved at isolere for en variabel x_j i en ligning, og derefter substituere udtrykket for x_j ind i de andre ligninger, kan en basal variabel udskiftes med en ikke-basal variabel, hvilket kaldes for et *basisskift*, og har til formål at øge den basale objektfunktion. Variablen, der indsættes i basis kaldes den *indgående variabel*, mens variablen der forlader basis, kaldes den *udgående variabel*. Konkordansformen er vigtig, da formen giver et overblik over hvilke variable, der er basale, og hvilke der er ikke-basale variable og dermed gør at basisskift bliver nemmere at udføre.

Eksempel 4.0.1. Betragt det lineære optimeringsproblem på standardform.

$$\begin{aligned} \text{Maksimér: } z = & \ 2x_1 - 3x_2 - 3x_3 \\ \text{betinget af: } & -3x_1 + 2x_2 \leq 80 \\ & -x_1 + x_2 + 4x_3 \leq 20 \\ & -2x_1 - 2x_2 + 5x_3 \leq 30 \\ & x_1, \quad x_2, \quad x_3 \geq 0 \end{aligned} \tag{4.3}$$

Dette problem er ubegrænset, da x_1 kan have en vilkårlig stor værdi uden at betingelserne bliver brudt. Det gælder generelt, at hvis der eksisterer et x_j så $c_j > 0$ og $a_{i,j} \leq 0$, for alle $i \in 1, \ldots, m$, så er problemet ubegrænset. Betragt det lineære optimeringsproblem på standardform.

$$\begin{aligned} \text{Maksimér: } z = & -2x_1 - 3x_2 - 3x_3 \\ \text{betinget af: } & \ 3x_1 + 2x_2 \leq 80 \\ & -x_1 + x_2 + 4x_3 \leq 20 \\ & \ 2x_1 - 2x_2 + 5x_3 \leq 30 \\ & x_1, \quad x_2, \quad x_3 \geq 0 \end{aligned} \tag{4.4}$$

Det følger af ikke-negativitetsbetingelserne, at maksimum bestemmes ved at sætte $x_1, x_2, x_3 = 0$. Det følger generelt, at hvis $\overrightarrow{c} \leq \overrightarrow{0}$ så er maksimum bestemt ved $\overrightarrow{x} = \overrightarrow{0}$, hvis dette er en mulig løsning. Den optimale løsning kan så bestemmes ved at sætte variablene i objektfunktionen til 0. Der ønskes nu en metode til at bestemme en løsning til et lineært programmeringsproblem ud fra denne observation. Betragt det lineære programmeringsproblem (4.5).

$$\begin{aligned} \text{Maksimér: } z = & \ 2x_1 + 3x_2 + 3x_3 \\ \text{betinget af: } & \ 3x_1 + 2x_2 \leq 80 \\ & -x_1 + x_2 + 4x_3 \leq 20 \\ & \ 2x_1 - 2x_2 + 5x_3 \leq 30 \\ & x_1, \quad x_2, \quad x_3 \geq 0 \end{aligned} \tag{4.5}$$

Da der eksisterer positive værdier i $\overrightarrow{c}$ vil det sige, at maksimum ikke kan bestemmes ved at sætte $x_1, x_2, x_3 = 0$. Derfor indføres en metode til at ændre koefficienter i objektfunktionen. Til dette formål skal problemet omformes til et problem på slackform:

$$\begin{aligned} \text{Maksimér: } z = & \ 2x_1 + 3x_2 + 3x_3 \\ \text{betinget af: } & \ 3x_1 + 2x_2 + x_4 = 80 \\ & -x_1 + x_2 + 4x_3 + x_5 = 20 \\ & \ 2x_1 - 2x_2 + 5x_3 + x_6 = 30 \\ & x_1, \quad x_2, \quad x_3, \quad x_4, \quad x_5, \quad x_6 \geq 0 \end{aligned} \tag{4.6}$$

Vi vil gerne erstatte variablen x_2 i objektfunktionen med en anden variabel. Maksimum bestemmes ved at foretage basisskift, indtil alle koefficienter i $\overrightarrow{c}$ er ikke-positive. Da x_2 og x_3 har samme koefficient i $\overrightarrow{c}$, som også er den største, så vælges x_2, da den har lavest indeks. En værdi af x_2 som er større end 0 vil derfor give en større værdi af objektfunktionen. Det bestemmes nu hvilken variabel, der skal skiftes ud. Til dette formål omskrives problemet til konkordansform.

$$
\begin{aligned}
\text{Maksimér: } & z = 2x_1 + 3x_2 + 3x_3 \\
\text{betinget af: } & 80 - 3x_1 - 2x_2 = x_4 \\
& 20 + x_1 - x_2 - 4x_3 = x_5 \\
& 30 - 2x_1 + 2x_2 - 5x_3 = x_6 \\
& x_i \geq 0 \quad (1 \leq i \leq 6)
\end{aligned} \tag{4.7}
$$

Nu observeres det at en stigning i værdien af x_2 vil medføre et fald i værdien af x_4 og x_5, men en stigning i værdien af x_6. Derfor skal det sikres, at værdien af x_4, x_5, x_6 ikke bliver negativ. Derved kan følgende uligheder opstilles, ved at sætte alle variable undtagen x_2 til nul, for at vi kan bestemme den maksimale værdi som x_2 kan antage, uden at de basale variables ikke-negativitetsbetingelser bliver brudt.

$$
0 \leq 80 - 2x_2 \qquad 0 \leq 20 - x_2 \qquad 0 \leq 30 + 2x_2 \tag{4.8}
$$

Derved ses det at $x_2 \leq 40$, $x_2 \leq 20$, $-15 \leq x_2$. Den sidste betingelse er overflødig grundet ikke-negativitetsbetingelsen. Nu vælges den mest restriktive af de to betingelser, hvilket er $x_2 \leq 20$. Der foretages basisskift så x_2 bliver en basal variabel og x_5 bliver en ikke-basal variabel.

$$
\begin{aligned}
\text{Maksimér: } & z = 60 + 5x_1 - 3x_5 - 9x_3 \\
\text{betinget af: } & 40 - 5x_1 + 2x_5 + 8x_3 = x_4 \\
& 20 + x_1 - x_5 - 4x_3 = x_2 \\
& 70 - 2x_5 - 12x_3 = x_6 \\
& x_i \geq 0 \quad (1 \leq i \leq 6)
\end{aligned} \tag{4.9}
$$

Nu skal x_1 erstattes, da den stadig har positiv koefficient og processen gentages indtil alle koefficienter i $\overrightarrow{c}$ er negative, da den optimale løsning så kan bestemmes. Antag nu, at alle de uligheder der var fundet i (4.8) havde vist at $x_2 \geq k$, hvor alle $k < 0$. Herved er x_2 ubegrænset, og derfor er konkordansen ubegrænset, og intet optimum kan findes.

4.1 Algoritmen

Nu opsummeres de forskellige skridt i algoritmen, forudsat at $\vec{b} \geq \vec{0}$. Hvis $\vec{b} < \vec{0}$, kan problemet stadig løses, hvilket vil blive beskrevet i Afsnit 4.3. Algoritmen tager som input en konkordans:

$$\begin{aligned} \text{Maksimér: } & z = v + \vec{c} \cdot \vec{x} \\ \text{betinget af: } & \vec{x}_\beta = \vec{b} - A \cdot \vec{x}_\rho \\ & \vec{x}_\rho, \vec{x}_\beta \geq \vec{0}, \end{aligned}$$

hvor A er koefficientmatricen til et ligningssystem med m ligninger, og n variable. Algoritmen returnerer en optimal løsning til det lineære programmeringsproblem.
Her defineres de forskellige skridt i simplexalgoritmen.

a) Hvis $\vec{c} \leq \vec{0}$ så er v maksimum bestemt og algoritmen stopper her.

b) Hvis der eksisterer et koordinat $c_k > 0$ i $\vec{c}$, og der for alle $a_{i,k}$ gælder at $a_{i,k} \leq 0$ så stopper algoritmen, da problemet er ubegrænset.

c) Hvis ikke, så bestem variablen med laveste koefficient i objektfunktionen, og kald denne variabel for x_j. Hvis der eksisterer to koordinater i $\vec{c}$ med samme værdi, vælg den med det laveste indeks, og bestem herefter følgende:

$$r = \text{argmin}_i \left\{ a_{i,j} : \; x_i \in \beta, \; \frac{b_i}{a_{i,j}} > 0 \right\}. \tag{4.10}$$

Ratioen bestemmer hvilken betingelse, der er skarpest, og dermed hvilken variabel, der skal skiftes. Foretag basisskift så x_j bliver en basal variabel, og den basale variabel i række r bliver en ikke-basal variabel. Gentag indtil enten betingelse a) eller b) er opfyldt.

Skridt a) og b) i simplexalgoritmen kaldes *afslutningskriterierne*, da en af disse skal være opfyldt, for at algoritmen slutter og returnerer et svar. Skridt c) er selve udførelsen; her foretages basisskift og en ny konkordans dannes.

Da den mindste ratio altid vælges, sikres det at den skarpeste begrænsning altid overholdes, og den nye løsning vil være mulig. Derudover skal det nævnes, at når der foretages et basisskift, og ligningen for den nye basale variabel indsættes i objektfunktionen, så lægges værdien af ligningens $\frac{b_i}{a_{i,j}}$ til objektfunktionens v. Eftersom at ratioen skal være ikke-negativ, så vil værdien af ligningens b også være ikke-negativ.

4.2 Tableaumatricer

I denne sektion indføres *tableaumatricer*, som gør simplexalgoritmen mere overskuelig. Lad en konkordans være givet:

$$\begin{aligned}
\text{Maksimér: } & z = v + \vec{c} \cdot \vec{x} \\
\text{betinget af: } & \vec{x}_\beta = \vec{b} - A \cdot \vec{x}_\rho \\
& \vec{x}_\rho, \vec{x}_\beta \geq \vec{0}.
\end{aligned}$$

En matrix, der indeholder alle koefficienter for de basale variable og alle koefficienter for ikke-basale variable, samt betingelsesvektoren $\vec{b}$ og objektvektoren $\vec{c}$, opstilles. Vektoren $\vec{R}_j$ er den vektor, der indeholder alle ratioerne $r_i = \frac{b_i}{a_{i,j}}$ for alle $i \in \beta$. Dette gør det hurtigere at sammenligne hvilken ratio, der er mindst. En tableaumatrix opskrives på følgende måde, hvor henholdsvis første, anden og tredje søjle indeholder koefficienterne til $\vec{x}_\rho$, $\vec{x}_\beta$ og z.

$$\left[\begin{array}{c|c|c|c} A & I_m & \vec{0} & \vec{b} \\ \hline -\vec{c} & \vec{0}^T & 1 & 0 \end{array}\right], \vec{R}_j. \tag{4.11}$$

Eksempel 4.2.1. Betragt det lineære programmeringsproblem på standardform.

$$\begin{aligned}
\text{Maksimér: } z = {} & 2x_1 - 3x_2 - 3x_3 \\
\text{betinget af: } & 3x_1 + 2x_2 \leq 60 \\
& -x_1 + x_2 + 4x_3 \leq 10 \\
& 2x_1 - 2x_2 + 5x_3 \leq 50 \\
& x_1, \quad x_2, \quad x_3 \geq 0.
\end{aligned} \tag{4.12}$$

Omskriv dernæst til slackform:

$$\begin{aligned}
\text{Maksimér: } z = {} & 2x_1 - 3x_2 - 3x_3 \\
\text{betinget af: } & 3x_1 + 2x_2 + x_4 = 60 \\
& -x_1 + x_2 + 4x_3 + x_5 = 10 \\
& 2x_1 - 2x_2 + 5x_3 + x_6 = 50 \\
& x_1, \quad x_2, \quad x_3, \quad x_4, \quad x_5, \quad x_6 \geq 0.
\end{aligned} \tag{4.13}$$

Nu opskrives tableaumatricen for problemet, og da c_1 er positiv, skal pivoteringen foretages for x_1. Herefter udregnes ratioerne for x_1 for hver række.

$$\left[\begin{array}{ccc|ccc|c|c} 3 & 2 & 0 & 1 & 0 & 0 & 0 & 60 \\ -1 & 1 & 4 & 0 & 1 & 0 & 0 & 10 \\ 2 & -2 & 5 & 0 & 0 & 1 & 0 & 50 \\ \hline -2 & 3 & 3 & 0 & 0 & 0 & 1 & 0 \end{array}\right], \begin{bmatrix} 20 \\ -10 \\ 25 \end{bmatrix}.$$

Da koefficienten for x_1 i r_2 er negativ, vil den ikke begrænse værdien af x_1. Den strengeste betingelse er i r_1, der begrænser x_1 til at være 20, og derfor foretages der en pivotering i $a_{1,1}$.

$$\left[\begin{array}{ccc|ccc|c|c} 1 & \frac{2}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 20 \\ 0 & \frac{5}{3} & 4 & \frac{1}{3} & 1 & 0 & 0 & 30 \\ 0 & -\frac{10}{3} & 5 & -\frac{2}{3} & 0 & 1 & 0 & 10 \\ \hline 0 & \frac{13}{3} & 3 & \frac{2}{3} & 0 & 0 & 1 & 40 \end{array}\right].$$

Da der ikke er flere negative c-værdier i tableaumatricen, er processen slut. Da der står $-\overrightarrow{c}$ i tableaumatricen, vil det betyde at alle c-værdierne er negative og der skal indsættes 0 for at fremlede den optimale løsning. Den sidste række er således givet ved $\frac{13}{3}x_2+3x_3+\frac{2}{3}x_4+z = 40$, da sættes alle de basale variable til 0 resulterende i at $z' = 40$. Problemet har nu den optimale løsning $\overrightarrow{x}' = [20, 0, 0]^T$, hvilket giver den maksimale værdi af $z' = 2 \cdot 20 - 3 \cdot 0 - 3 \cdot 0 = 40$. Det kan undersøges om løsningen er mulig ved at indsætte den i problemet på standardform.

$$\begin{aligned} \text{Maksimér: } z = \quad & 2 \cdot 20 - 3 \cdot 0 - 3 \cdot 0 = 40 \\ \text{betinget af: } \quad & 3 \cdot 20 + 2 \cdot 0 = 60 \leq 60 \\ & -1 \cdot 20 + 1 \cdot 0 + 4 \cdot 0 = -20 \leq 10 \\ & 2 \cdot 20 - 2 \cdot 0 + 5 \cdot 0 = 40 \leq 50. \end{aligned} \tag{4.14}$$

Derved kan det ses, at alle betingelser er overholdt.

4.3 Assisterende metode

Denne undersektion er udarbejdet ud fra [Israel, 2006]. For problemer, hvor en eller flere af indgangene i $\overrightarrow{b}$ er negative, er det ikke muligt at sætte de basale variable til $\overrightarrow{b}$, da det bryder ikke-negativitetsbetingelserne. Nu beskrives der en metode til først at bestemme om et problem er muligt, og derefter til at omskrive problemet således, at en mulig løsning i den sidste konkordans er $\overrightarrow{x}_\beta = \overrightarrow{b}$ og $\overrightarrow{x}_\rho = \overrightarrow{0}$. Til at undersøge om et problem på standardform, som i denne sammenhæng kaldes for *standardproblemet*, har en mulig løsning, opstilles et *assisterende problem*. Det assisterende problem har de samme begrænsninger som standardproblemet, men derudover subtraheres der en *artificiel variabel*, der kaldes

x_0, fra alle ligninger med en negativ b-værdi. Yderligere er det assisterende problems objektfunktion $w = -x_0$, hvilket er opstillet på generel form i Ligning (4.16).

Standardproblem

$$\text{Maksimér: } z = \sum_{j=1}^{n} c_j \cdot x_j \tag{4.15}$$
$$\text{betinget af: } \sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i$$
$$x_j \geq 0.$$

Assisterende problem

$$\begin{aligned} &\text{Maksimér: } w = -x_0 \\ &\text{betinget af: } && \sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i (hvis\ \ b_i \geq 0) \\ & && -x_0 + \sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i (hvis\ \ b_i < 0) \\ & && x_0, \qquad x_j \geq 0 \end{aligned} \tag{4.16}$$

Det vises nu, at hvis $w' \neq 0$, så har standardproblemet ikke en løsning.

Sætning 4.3.1. *Lad P være et lineært optimeringsproblem, hvis løsning er ikke-mulig, og lad Q være P's assisterende problem. Så er P muligt hvis og kun hvis Q er optimal i 0.*

Bevis. Da dette er en biimplikation, så deles sætningen nu op i to. Først bevises, at hvis P er muligt, så gælder det om Q's objektfunktion w, at $w' = 0$. Lad nu P være muligt. Så eksisterer der $\overrightarrow{x} \geq \overrightarrow{0}$, som opfylder $A \cdot \overrightarrow{x} \leq \overrightarrow{b}$. Da dette kan overholdes, så er $x_0 = 0$ også muligt i Q og derved er $w' = 0$, da alle andre x_0 værdier skaber mindre w. Lad nu P være ikke-muligt, så kan $A \cdot \overrightarrow{x} \leq \overrightarrow{b}$ ikke overholdes for nogen $\overrightarrow{x} \geq \overrightarrow{0}$, og derved er $x_0 = 0$ ikke-muligt, og $w' < 0$. ■

Nu beskrives trinene for den assisterende metode:

a) Omskriv det assisterende problem til slackform.

$$\begin{aligned} &\text{Maksimér: } w = -x_0 \\ &\text{betinget af: } && \sum_{j=1}^{n} a_{i,j} \cdot x_j + s_i = b_i (b_i > 0) \\ & && -x_0 + \sum_{j=1}^{n} a_{i,j} \cdot x_j + s_i = b_i (b_i < 0) \\ & && x_0, \qquad x_j,\ s_i \geq 0. \end{aligned} \tag{4.17}$$

b) Omskriv til en konkordans og opstil tableaumatricen for konkordansen. Det assisterende problems tableaumatrix opstilles på en særlig måde, for senere at kunne omskrive den til standardproblemets tableaumatrix. Den dannes som udgangspunkt normalt, men der tilføjes en række med standardproblemets objektfunktion, og en søjle tilhørende koefficienterne for z.

$$\left[\begin{array}{c|c|c|c|c|c} \overrightarrow{d} & A & I_m & \overrightarrow{0} & \overrightarrow{0} & \overrightarrow{b} \\ \hline 0 & -\overrightarrow{c} & \overrightarrow{0}^T & 1 & 0 & 0 \\ \hline 1 & \overrightarrow{0}^T & \overrightarrow{0}^T & 0 & 1 & 0 \end{array}\right]. \tag{4.18}$$

Vektoren $\overrightarrow{d}$, som indeholder koefficienterne for x_0, har indgangen 0 i rækker, hvor $b_i \geq 0$, og -1 i rækker hvor $b_i < 0$. Desuden ignoreres rækken med standardproblemets objektfunktion, når der bestemmes, hvor der skal pivoteres.

c) Pivotér x_0 i rækken med den laveste b-værdi. Dermed bliver $\overrightarrow{b} \geq \overrightarrow{0}$. Dette sker da rækken med den laveste b-værdi ganges igennem med -1, og derefter lægges til alle andre rækker med negative værdier for b.

d) Benyt simplexalgoritmen til at løse det assisterende problem. Hvis $w' < 0$ har standardproblemet ikke en løsning. Hvis $w' = 0$ har standardproblemet en løsning, og det omskrives derfor til en konkordans og opstilles som en tableaumatrix.

e) Samme rækkeoperationer, der løste det assisterende problem, udføres nu på konkordansens tableaumatrix, hvilket resulterer i udelukkende ikke-negative værdier af b, hvilket medfører at alle konkordansers løsninger er mulige.

f) Herfra benyttes simplexalgoritmen til at løse standardproblemet.

Skridt d) og e) kan gøres hurtigere ved at fjerne søjlerne med koefficienterne for x_0 og w, samt rækken med objektfunktion fra tableaumatricen til det assisterende problem.

Eksempel 4.3.2. Det ønskes at optimere problemet.

$$\begin{array}{llrcrcrcr} \text{Maksimér:} & z = & 17x_1 & + & 70x_2 & + & 13x_3 & & \\ \text{betinget af:} & & 2x_1 & - & 6x_2 & - & 4x_3 & \leq & -4 \\ & & -1x_1 & - & 2x_2 & - & 3x_3 & \leq & -2 \\ & & 4x_1 & + & 2x_2 & - & 1x_3 & \leq & 3 \\ & & -3x_1 & + & 3x_2 & - & 3x_3 & \geq & 1 \\ & & x_1, & & x_2, & & x_3 & \geq & 0. \end{array} \tag{4.19}$$

Først ganges fjerde begrænsning igennem med -1 så ulighedstegnet vendes.

$$
\begin{aligned}
\text{Maksimér: } & z = 17x_1 + 70x_2 + 13x_3 \\
\text{betinget af: } & 2x_1 - 6x_2 - 4x_3 \leq -4 \\
& -1x_1 - 2x_2 - 3x_3 \leq -2 \\
& 4x_1 + 2x_2 - 1x_3 \leq 3 \\
& 3x_1 - 3x_2 + 3x_3 \leq -1 \\
& x_1, \quad x_2, \quad x_3 \geq 0.
\end{aligned}
\tag{4.20}
$$

Det assisterende problem opstilles på slackform. Dette er nødvendigt, da mindst én af koordinaterne i $\overrightarrow{b}$ er negativ, og løsningen $\overrightarrow{x}_\rho = \overrightarrow{0}$ er derfor ikke mulig.

$$
\begin{aligned}
\text{Maksimér: } & w = -x_0 \\
\text{betinget af: } & -x_0 + 2x_1 - 6x_2 - 4x_3 + x_4 \leq -4 \\
& -x_0 - 1x_1 - 2x_2 - 3x_3 + x_5 \leq -2 \\
& 4x_1 + 2x_2 - 1x_3 + x_6 \leq 3 \\
& -x_0 + 3x_1 - 3x_2 + 3x_3 + x_7 \leq -1 \\
& x_0, \quad x_1, \quad x_2, \quad x_3, \; x_4, \; x_5, \; x_6, \; x_7 \geq 0.
\end{aligned}
\tag{4.21}
$$

Dette omskrives til en konkordans.

$$
\begin{aligned}
\text{Maksimér: } & w = -x_0 \\
\text{betinget af: } & x_4 = -4 + x_0 - 2x_1 + 6x_2 + 4x_3 \\
& x_5 = -2 + x_0 + 1x_1 + 2x_2 + 3x_3 \\
& x_6 = 3 - 4x_1 - 2x_2 + 1x_3 \\
& x_7 = -1 + x_0 - 3x_1 + 3x_2 - 3x_3 \\
& x_i \geq 0 (1 \leq i \leq 7).
\end{aligned}
\tag{4.22}
$$

Tableaumatricen for det assisterende problem opstilles.

$$
\left[\begin{array}{c|ccc|cccc|c|c|c}
-1 & 2 & -6 & -4 & 1 & 0 & 0 & 0 & 0 & 0 & -4 \\
-1 & -1 & -2 & -3 & 0 & 1 & 0 & 0 & 0 & 0 & -2 \\
0 & 4 & 2 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 3 \\
-1 & 3 & -3 & 3 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\
\hline
0 & -17 & -70 & -13 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right].
$$

Den laveste b-værdi er i første række, og derfor pivoteres der i $a_{1,1}$.

$$\left[\begin{array}{c|ccc|cccc|c|c|c}
1 & -2 & 6 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 4 \\
0 & -3 & 4 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 2 \\
0 & 4 & 2 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 3 \\
0 & 1 & 3 & 7 & -1 & 0 & 0 & 1 & 0 & 0 & 3 \\
\hline
0 & -17 & -70 & -13 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 2 & -6 & -4 & 1 & 0 & 0 & 0 & 0 & 1 & -4
\end{array}\right].$$

Ratioerne for x_2 bestemmes til at være $\overrightarrow{R} = [\frac{2}{3}, \frac{1}{2}, \frac{3}{2}, 1]^T$, og der pivoteres i $a_{2,3}$.

$$\left[\begin{array}{c|ccc|cccc|c|c|c}
1 & \frac{5}{2} & 0 & \frac{5}{2} & \frac{1}{2} & -\frac{3}{2} & 0 & 0 & 0 & 0 & 1 \\
0 & -\frac{3}{4} & 1 & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & \frac{11}{2} & 0 & -\frac{3}{2} & \frac{1}{2} & -\frac{1}{2} & 1 & 0 & 0 & 0 & 2 \\
0 & \frac{13}{4} & 0 & \frac{25}{4} & -\frac{1}{4} & -\frac{3}{4} & 0 & 1 & 0 & 0 & \frac{3}{2} \\
\hline
0 & -\frac{139}{2} & 0 & \frac{9}{2} & -\frac{35}{2} & \frac{35}{2} & 0 & 0 & 1 & 0 & 35 \\
\hline
0 & -\frac{5}{2} & 0 & -\frac{5}{2} & -\frac{1}{2} & \frac{3}{2} & 0 & 0 & 0 & 1 & -1
\end{array}\right].$$

Herefter skal der nu pivoteres så x_1 bliver en basal variabel og ratioerne udregnes $\overrightarrow{R} = [\frac{2}{5}, -\frac{2}{3}, \frac{4}{11}, \frac{6}{13}]^T$ og der pivoteres i $a_{3,2}$.

$$\left[\begin{array}{c|ccc|cccc|c|c|c}
1 & 0 & 0 & \frac{35}{11} & \frac{3}{11} & -\frac{14}{11} & -\frac{5}{11} & 0 & 0 & 0 & \frac{1}{11} \\
0 & 0 & 1 & \frac{1}{22} & -\frac{2}{11} & \frac{2}{11} & \frac{3}{22} & 0 & 0 & 0 & \frac{17}{22} \\
0 & 1 & 0 & -\frac{3}{11} & \frac{1}{11} & -\frac{1}{11} & \frac{2}{11} & 0 & 0 & 0 & \frac{4}{11} \\
0 & 0 & 0 & \frac{157}{22} & -\frac{6}{11} & -\frac{5}{11} & -\frac{13}{22} & 1 & 0 & 0 & \frac{7}{22} \\
\hline
0 & 0 & 0 & -\frac{159}{11} & -\frac{123}{11} & \frac{123}{11} & \frac{139}{11} & 0 & 1 & 0 & \frac{663}{11} \\
\hline
0 & 0 & 0 & -\frac{35}{11} & -\frac{3}{11} & \frac{14}{11} & \frac{5}{11} & 0 & 0 & 1 & -\frac{1}{11}
\end{array}\right].$$

Der skal pivoteres i $a_{1,4}$, da x_3 skal blive en del af basis, og dens ratioer er $\overrightarrow{R} = [\frac{1}{35}, 17, -\frac{4}{3}, \frac{7}{157}]^T$.

$$\left[\begin{array}{c|ccc|cccc|c|c|c}
\frac{11}{35} & 0 & 0 & 1 & \frac{3}{35} & -\frac{2}{5} & -\frac{1}{7} & 0 & 0 & 0 & \frac{1}{35} \\
-\frac{1}{70} & 0 & 1 & 0 & -\frac{13}{70} & \frac{1}{5} & \frac{1}{7} & 0 & 0 & 0 & \frac{27}{35} \\
\frac{3}{35} & 1 & 0 & 0 & \frac{4}{35} & -\frac{1}{5} & \frac{1}{7} & 0 & 0 & 0 & \frac{13}{35} \\
-\frac{157}{70} & 0 & 0 & 0 & -\frac{81}{70} & \frac{12}{5} & \frac{3}{7} & 1 & 0 & 0 & \frac{4}{35} \\
\hline
\frac{159}{35} & 0 & 0 & 0 & -\frac{348}{35} & \frac{27}{5} & \frac{74}{7} & 0 & 1 & 0 & \frac{2124}{35} \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right].$$

Da $w' = 0$ har standardproblemet en løsning, og tableaumatricen omskrives, hvilket gøres ved at fjerne søjlerne 1 og 10 samt række 6.

$$\left[\begin{array}{ccc|cccc|c|c} 0 & 0 & 1 & \frac{3}{35} & -\frac{2}{5} & -\frac{1}{7} & 0 & 0 & \frac{1}{35} \\ 0 & 1 & 0 & -\frac{13}{70} & \frac{1}{5} & \frac{1}{7} & 0 & 0 & \frac{27}{35} \\ 1 & 0 & 0 & \frac{4}{35} & -\frac{1}{5} & \frac{1}{7} & 0 & 0 & \frac{13}{35} \\ 0 & 0 & 0 & -\frac{81}{70} & \frac{12}{5} & \frac{3}{7} & 1 & 0 & \frac{4}{35} \\ \hline 0 & 0 & 0 & -\frac{348}{35} & \frac{27}{5} & \frac{74}{7} & 0 & 1 & \frac{2421}{35} \end{array}\right].$$

Ratioerne for x_4 er $\overrightarrow{R} = [\frac{1}{3}, -\frac{54}{13}, \frac{13}{4}, -\frac{8}{81}]^T$, og der pivoteres i $a_{1,4}$.

$$\left[\begin{array}{ccc|cccc|c|c} 0 & 0 & \frac{35}{3} & 1 & -\frac{14}{3} & -\frac{5}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 1 & \frac{13}{6} & 0 & -\frac{2}{3} & -\frac{1}{6} & 0 & 0 & \frac{5}{6} \\ 1 & 0 & -\frac{4}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 0 & \frac{27}{2} & 0 & -3 & -\frac{3}{2} & 1 & 0 & \frac{1}{2} \\ \hline 0 & 0 & 116 & 0 & -41 & -6 & 0 & 1 & 64 \end{array}\right].$$

Der skal pivoteres i $a_{3,5}$ da ratioerne for x_5 er $\overrightarrow{R} = [-\frac{1}{14}, -\frac{5}{4}, 1, -\frac{1}{6}]^T$.

$$\left[\begin{array}{ccc|cccc|c|c} 14 & 0 & -7 & 1 & 0 & 3 & 0 & 0 & 5 \\ 2 & 1 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{3}{2} \\ 3 & 0 & -4 & 0 & 1 & 1 & 0 & 0 & 1 \\ 9 & 0 & \frac{2}{3} & 0 & 0 & \frac{3}{2} & 1 & 0 & \frac{7}{2} \\ \hline 123 & 0 & -48 & 0 & 0 & 35 & 0 & 1 & 105 \end{array}\right].$$

Nu udregnes ratioerne for x_3, $\overrightarrow{R} = [-\frac{5}{7}, -3, -\frac{1}{4}, \frac{7}{3}]^T$, og der pivoteres i $a_{4,3}$.

$$\left[\begin{array}{ccc|cccc|c|c} 56 & 0 & 0 & 1 & 0 & 10 & \frac{14}{3} & 0 & \frac{64}{3} \\ 5 & 1 & 0 & 0 & 0 & 1 & \frac{1}{3} & 0 & \frac{8}{3} \\ 27 & 0 & 0 & 0 & 1 & 5 & \frac{8}{3} & 0 & \frac{31}{3} \\ 6 & 0 & 1 & 0 & 0 & 1 & \frac{2}{3} & 0 & \frac{7}{3} \\ \hline 411 & 0 & 0 & 0 & 0 & 83 & 32 & 1 & 217 \end{array}\right].$$

Problemet har derfor løsningen $x_1 = 0, x_2 = \frac{8}{3}, x_3 = \frac{7}{3}$, hvilket giver værdien 217 for objektfunktionen.

4.4 Stærk Dualitet

Betragt igen primalformen (p) og dualformen(d) fra Afsnit 3.2 givet ved:

Primalform

$$\begin{aligned} \text{Maksimér: } & z = \vec{c} \cdot \vec{x} \\ \text{betinget af: } & A \cdot \vec{x} \leq \vec{b} \\ & \vec{x} \geq \vec{0}, \end{aligned} \tag{4.23}$$

Dualform

$$\begin{aligned} \text{Minimér: } & w = \vec{b}^T \cdot \vec{y} \\ \text{betinget af: } & A^T \cdot \vec{y} \geq \vec{c}^T \\ & \vec{y} \geq \vec{0}. \end{aligned} \tag{4.24}$$

I Sætning 3.10 blev svag dualitet bevist, og dermed er $z' \leq w'$ gældende. Nu vises det, at $z' = w'$, hvilket kaldes stærk dualitet.

Sætning 4.4.1. *Hvis et muligt begrænset lineært programmeringsproblem, p, på primalform har en optimal løsning z', så har den tilsvarende dual, d, en optimal løsning w', hvorom det gælder at $w' = z'$.*

Bevis. Dette bevis tager udgangspunkt i [Camarena, 2016]. Da det vides, at for alle løsninger til p og d, så er $z \leq w$, så er det nok at finde et muligt w, hvor $w = z$, for et muligt z. For at vise dette, så benyttes simplexalgoritmen nu på primalen. Det ønskes nu at inkludere alle variable og koefficienter, der optræder i en konkordans. Her kan et programmeringsproblem nu opstilles.

$$\begin{aligned} \text{Maksimér: } & z = \bar{c} \cdot \bar{x} \\ \text{betinget af: } & \bar{A} \cdot \bar{x} = \vec{b} \\ & \bar{x} \geq \vec{0}. \end{aligned} \tag{4.25}$$

Nu er $\bar{x} = [\vec{x}_\rho^T, \vec{x}_\beta^T]^T$ en vektor, der indeholder alle basale og ikke-basale variable. Matricen $\bar{A}$ er en augmenteret matrix af A_β og A_ρ, hvor A_β er en undermatrix af $\bar{A}$ med alle søjler, der repræsenterer de basale variable s koefficienter i algoritmens sidste konkordans, mens A_ρ er undermatricen med søjlerne, der repræsenterer de ikke-basale variables koefficienter i samme konkordans. Ligeledes er $\bar{c} = [\vec{c}, \vec{0}^T]$. Da $\bar{A}\bar{x} = \vec{b}$, opstilles udtrykket:

$$A_\beta \vec{x}_\beta + A_\rho \vec{x}_\rho = \vec{b}. \tag{4.26}$$

Gennem simplexalgoritmen forekommer der rækkereduktioner på A_β indtil sidste konkordans, hvori $A_\beta = I_m$, og da er A_β en invertibel matrix, bemærk at $\bar{A}$ også kan skrives som $\left[A \mid I_m\right]$. Her findes basisvariablene $\vec{x}_\beta$ som:

$$\vec{x}_\beta = A_\beta^{-1} \vec{b} - A_\beta^{-1} A_\rho \vec{x}_\rho. \tag{4.27}$$

Der opstilles et udtryk for objektfunktionen i algoritmens sidste konkordans, hvor ρ og β beskriver indeksmængden for den ikke-basale mængde og basis for den sidste konkordans.

$$\begin{aligned} z &= \bar{c} \cdot \bar{x} \\ &= \vec{c}_\beta \cdot \vec{x}_\beta + \vec{c}_\rho \cdot \vec{x}_\rho \\ &= \vec{c}_\beta (A_\beta^{-1} \vec{b} - A_\beta^{-1} A_\rho \vec{x}_\rho) + \vec{c}_\rho \cdot \vec{x}_\rho \\ &= \vec{c}_\beta A_\beta^{-1} \vec{b} + (\vec{c}_\rho - \vec{c}_\beta A_\beta^{-1} A_\rho) \vec{x}_\rho. \end{aligned} \tag{4.28}$$

Da dette er fra algoritmens sidste konkordans, er den objektfunktion, hvor $\vec{x}_\rho = \vec{0}$ også den optimale objektfunktion , og derved er

$$\vec{c}_\rho - \vec{c}_\beta A_\beta^{-1} A_\rho \leq \vec{0}. \tag{4.29}$$

Hermed er der en ligning for den j'te indgang i $\vec{c}_\rho$, hvor $\vec{a}_j$ er den j'te søjle i A_ρ:

$$c_j - \vec{c}_\beta A_\beta^{-1} \vec{a}_j \leq 0, j \in \rho. \tag{4.30}$$

Ligeledes kan det findes at denne ligning er gældende for alle $j \in \beta$. Hvis de basale variable indsættes, så bestemmes det at:

$$\vec{c}_\beta - \vec{c}_\beta A_\beta^{-1} A_\beta \leq \vec{0}, \tag{4.31}$$

hvilket er gældende. Derved er ulighed (4.30) gældende for alle $j \in \beta \cup \rho$. I det originale optimeringsproblem, og dermed den første konkordans, kaldes de basale og de ikke-basale variable for henholdsvis $\vec{x}_{\beta_0}$ og $\vec{x}_{\rho_0}$. Så er $A_{\beta_0} = I_m$, $\vec{c}_{\beta_0} = \vec{0}$, $\vec{c}_{\rho_0} = \vec{c}$ og $A_{\rho_0} = A$. Derved kan uligheden (4.30) opdeles i $\vec{x}_{\beta_0}$ og $\vec{x}_{\rho_0}$. Dette producerer de følgende to uligheder;

$$\vec{c}_{\rho_0} - \vec{c}_\beta A_\beta^{-1} A_{\rho_0} = \vec{c} - \vec{c}_\beta A_\beta^{-1} A \leq \vec{0}, \tag{4.32}$$

$$\vec{c}_{\beta_0} - \vec{c}_\beta A_\beta^{-1} A_{\beta_0} = \vec{0} - \vec{c}_\beta A_\beta^{-1} I_m \leq \vec{0}. \tag{4.33}$$

Nu vises, at $\vec{y}^T = \vec{c}_\beta A_\beta^{-1}$ er en løsning til dualen. Dualen er betinget af $A^T \vec{y} \geq \vec{c}^T$. Ud fra dette, kan det findes at:

$$\begin{aligned} \vec{c} - \vec{c}_\beta A_\beta^{-1} A &\leq \vec{0} \\ \vec{c}^T - A^T (A_\beta^{-1})^T \vec{c}_\beta^T &\leq \vec{0}^T \\ A^T \vec{y} &\geq \vec{c}^T. \end{aligned} \tag{4.34}$$

Derved overholdes dualens betingelser. Ikke-negativitetsbetingelsen skal $\vec{y}$ også overholde. Hvis $\vec{y} = \vec{c}_\beta A_\beta^{-1}$ indsættes i uligheden (4.33), får man $-\vec{y} \leq \vec{0}$, hvilket betyder at $\vec{y}$ overholder ikke-negativitesbetingelsen. Tilsidst skal det vises, at $\vec{b}^T \cdot \vec{y} = \vec{c} \cdot \vec{x}'$. Bemærk her, at $\vec{b}^T \cdot \vec{y} = \vec{y}^T \cdot \vec{b} = (\vec{c}_\beta \cdot A_\beta^{-1}) \cdot \vec{b}$. Ud fra Ligning (4.28), så kan det netop ses, at $(\vec{c}_\beta \cdot A_\beta^{-1}) \cdot \vec{b} = z'$, da $\vec{x}_\rho = 0$ for den optimale løsning, hvilket betyder at $\vec{b}^T \cdot \vec{y} = z'$. Derved er der en løsning dualen og en i primalen, der har samme objektfunktionsværdi, så ud fra svag dualitet er $w' = z'$, og stærk dualitet er hermed bevist. ■

4.5 Degenerering

En konkordans D med en basis $\vec{x}_\beta$, hvorom det gælder for alle $i \in \beta$ at $x_i \neq 0$, kaldes *ikke-degenereret* og løsningen $\vec{x}_\rho = \vec{0}$ kaldes også ikke-degenereret, og derfor stiger værdien af D's basale objektfunktionen fra den forrige konkordans. Hvis en konkordans er ikke-degenereret, betyder det at algoritmen nærmer sig den optimale løsning efter pivoteringen der ledte op til konkordansen. Derved vil et lineært optimeringsproblem, der kun har ikke-degenererede konkordanser tilnærme sig den optimale løsning ved hver iteration. Det vides, at algoritmen terminerer, når der ikke opstår *cykler*, idet der kun findes et begrænset antal basiser, og derfor vil løsningen blive bestemt. Hvis der derimod er en konkordans, der er *degenereret*, så er det muligt, at den næste iteration af algoritmen ikke sikrer en større værdi af den basale objektfunktion, for løsningen $\vec{x}_\rho = \vec{0}$, og derved laver algoritmen ikke fremskridt. Hvis basisskiftet foretages i en række, hvor det konstante led er 0, så betyder det netop at v ikke ændres, og derved stiger værdien af objektfunktionen ikke. Man kan se, at algoritmen vil degenerere, når man bestemmer ratioerne og der er to af ratioerne der er mindst. Degenerering sker, når man har en overflødig begrænsning, men dette vises ikke i projektet. En algoritme kan stadig terminere trods degenerering, men det er uønsket, da der kan opnås iterationer i algoritmen, hvor alle konkordanser er degenererede, hvilket betyder, at der ikke opnås fremskridt, og algoritmen vil begynde at cykle. Hvis dette er tilfældet, vil algoritmen aldrig terminere. Hermed skal cykler analyseres, når det kommer til at beslutte, om algoritmen terminerer. Her er der taget udgangspunkt i [Newman, 2016], [CS-149 Staff, 2007] og [Math. Dept., 2016].

Definition 4.5.1. *Hvis simplexalgoritmen skaber en konkordansfølge $K = \{\dots, D_1, D_2, \dots, D_k, D_1, \dots\}$, siges der at opstå en cykel, og simplexalgoritmen kan ikke terminere.*

Når der forekommer cykler, vil den basale objektfunktion ikke stige, og algoritmen vil ikke nærme sig den optimale objektfunktion.

Lemma 4.5.2. *Lad D_i og D_{i+1} være konkordanser med samme værdi for den basale objektfunktion, da har de to konkordanser samme løsning.*

Bevis. Betragt konkordansen D_i i en cykel, hvor $\vec{x}_{\beta i}$ er basis og de ikke-basale variable er $\vec{x}_{\rho i}$. I D_i er x_e den indgående variabel, og x_l er den udgående variabel.

$$\begin{aligned} x_j &= b_j - \sum_{i \in \rho_i \setminus \{e\}} a_{j,i} \cdot x_i - a_{j,e} \cdot x_e, \quad j \in \beta_i \setminus \{l\} \\ x_l &= b_l - \sum_{i \in \rho_i \setminus \{e\}} a_{l,i} \cdot x_i - a_{l,e} \cdot x_e, \\ z &= v + \sum_{i \in \rho_i \setminus \{e\}} c_i \cdot x_i + c_e \cdot x_e. \end{aligned} \tag{4.35}$$

Da det er x_e som er den indgående variabel og x_l, som er udgående, vil det sige at $c_e > 0$ og $a_{l,e} > 0$. Derved må det gælde, at $b_l = 0$, hvilket medfører at $x_l = 0$, for ellers vil v stige i værdi i den næste konkordans. Betragt nu konkordans D_{i+1} med den nye basis $\vec{x}_{\beta_{i+1}}$ og nye ikke-basale variable $\vec{x}_{\rho_{i+1}}$.

$$\begin{aligned} x_j &= b_j - \sum_{i \in \rho_i \setminus \{e\}} a_{j,i} \cdot x_i + a_{j,e} \cdot \sum_{i \in \rho_{i+1}} \hat{a}_{l,i} \cdot x_i, \quad j \in \beta_i \setminus \{l\} \\ x_e &= 0 - \sum_{i \in \rho_{i+1}} \hat{a}_{l,i} \cdot x_i \\ z &= v + \sum_{i \in \rho_i \setminus \{e\}} c_i \cdot x_i + c_e \cdot \left(- \sum_{i \in \rho_{i+1}} \hat{a}_{l,i} \cdot x_i\right). \end{aligned} \tag{4.36}$$

Derved må der gælde: $x_j = b_j$ i D_i og $x_j = b_j$ i D_{i+1}. Det vil sige værdien af x_j ikke ændres fra D_i til D_{i+1} og derved er løsning den samme. ■

4.5.1 Blands Algoritme

Blands algoritme sørger for at simplexalgoritmen ikke degenererer, ved at gå gennem variablene i leksikografisk orden. For en given konkordans er objektfunktionen $z = v + \sum_{j \in \rho} c_j x_j$. Nu bliver den indgående variabel valgt som et koordinat x_e i $\vec{x}_\rho$, hvor $c_e > 0$ og $e \le j$ for alle j hvor $c_j > 0$. Den udgående variabel bliver valgt som koordinatet x_{i_0} i $\vec{x}_\beta$, hvor

$$i_0 = \operatorname{argmin}_i \left\{ \frac{b_i}{a_{i,e}} : \ i \in \beta, \ a_{i,e} > 0 \right\}, \tag{4.37}$$

og $i_0 \le i$ når $\frac{b_{i_0}}{a_{i_0,e}} = \frac{b_i}{a_{i,e}}$, for i, sådan at $i \in \beta$.

Sætning 4.5.3. *Simplexalgoritmen terminerer og finder den optimale løsning, når Blands algoritme anvendes.*

Bevis. Sætingen bevises ved modstid: Lad $K = \{D_1, D_2, \dots, D_k, D_1, \dots\}$ være en cykel i simplexalgoritmen. Alle konkordanserne i følgen K har nu samme løsning, givet ud fra

Lemma 4.5.2. Hvis en variabel forlader en basis i en iteration i cyklen, må den returnere til basis i en anden iteration. Variable, der forlader en basis kaldes nu *ubestandige*. Mængden af ubestandige variable benævnes med $\mathcal{F}$. Alle ubestandige variable må tage værdien nul i løsningen til K, da de i en af konkordanserne er en ikke-basal variabel. Lad x_l være en variabel i $\mathcal{F}$, således l er størst mulig. Der er nu en konkordans D, hvor l forlader den basale mængde, og e forlader den ikke-basale mængde. Da l var størst mulig, så er $l > e$. Her er D på formen.

$$\begin{aligned} \text{Maksimér: } z =& v + \sum_{j\in\rho} c_j x_j \qquad (4.38)\\ \text{betinget af:} \quad & x_i = b_i - \sum_{j\in\rho} a_{i,j} x_j (\forall i \in \beta)\\ & x_j \geq 0(\forall i \in \beta). \end{aligned}$$

Lad $D^* \in K$ være den konkordans i cyklen, hvor x_l genindtræder i basis. Nu kan en objektfunktion for D^* opskrives.

$$z = v + \sum_{j=1}^{|\beta|+|\rho|} c_j^* x_j \tag{4.39}$$

Her er lighedsbetingelserne i $c_j^* = 0$ for $j \in \beta^*$. Per Lemma 4.5.2 så har v samme værdi i D og D^*, da $D, D^* \in K$. D's betingelser er gældende for alle mulige løsninger, og derved bestemmes nu en løsning for D, hvor $x_e = t$, $x_j = 0$ for alle $j \in \rho$ og $j \neq e$. Her er

$$x_i = b_i - a_{i,e}t(\forall i \in \beta). \tag{4.40}$$

Fra D og D^* bestemmes nu

$$z = v + c_e t \tag{4.41}$$

$$z = v + c_e^* t + \sum_{i\in\beta} c_i^* (b_i - a_{i,e}t). \tag{4.42}$$

Ud fra dette bestemmes det at

$$c_e t = c_e^* t + \sum_{i\in\beta} c_i^* (b_i - a_{i,e}t) \tag{4.43}$$

$$t(c_e - c_e^* + \sum_{i\in\beta} c_i^* a_{i,e}) = \sum_{i\in\beta} c_i^* b_i. \tag{4.44}$$

Dette er nu gældende for alle $t \in \mathbb{R}$. Da højresiden er konstant, så må den være nul, og

$$c_e - c_e^* + \sum_{i\in\beta} c_i^* a_{i,e} = 0. \tag{4.45}$$

Da x_e er den indgående variabel i D, så er $c_e > 0$. Da $e < l$ og x_l er indgående i D^*, er $c_e^* \leq 0$ per Blands algoritme. Derved er $c_e - c_e^* > 0$, og så er $\sum_{i\in\beta} c_i^* a_{i,e} < 0$. Herved er der et $b \in \beta$, sådan at $c_b^* a_{b,e} < 0$. Da x_b er basal i D, og $c_b^* \neq 0$, så er x_b ikke-basal i D^*. Herved er $x_b \in \mathcal{F}$, og $b \leq l$ per antagelse. Da x_l er den indgående variabel i D^*, så er $c_l^* > 0$. Fordi x_e er den indgående variabel i D, og x_l er den udgående variabel, så er $a_{l,e} > 0$. Altså er $c_l^* a_{l,e} > 0$. Da $c_b^* a_{b,e} < 0$, så bestemmes det, at $b < l$. Ud fra bestemmes findes det, at b ikke er en mulig kandidat for indgående variabel i D^*, altså er $c_b^* < 0$, men herfra er $a_{b,e} > 0$. I løsningen $\overrightarrow{x}_\rho = \overrightarrow{0}$ er $x_b = 0$, men da dette er gældende, og $a_{b,e} > 0$, så må x_b være en kandidat for den udgående variabel i D, men x_l forlader D. Dette er ikke muligt per Blands algoritme, og der er herved modstrid. ■

4.6 Kompleksitet

Problemet med Blands algoritme er, hvor langsom den er i forhold til andre pivotregler. Fordelen ved ikke at danne cykler er ikke så nødvendigt i praksis, da cykler sjældent forekommer. Derfor anvendes andre hurtigere pivotregler. At bevise deres kompleksitet er ud over dette projekts rammer, men det er stadig vigtigt at redegøre for de forskellige pivotreglers kompleksitet. Det mest anvendte bevis for simplexalgoritmens kompleksitet stammer fra Klee og Minty, som anvendte *Klee-Minty kasser* til at vise at rigtig mange pivotregler har en kompleksitet på $O(2^n)$, hvor n er antal variable. Dette er ikke et fantastisk resultat, da den viser at worstcase-kompleksiteten for simplexalgoritmen er eksponentiel. Det er dog vist at Blands algoritme har en kompleksitet på $O(e^{C \cdot \sqrt{n \, ln(n)}})$, hvor C er en konstant. Dette er bedre end $O(2^n)$, men stadig langt fra polynomiel kompleksitet. *Hirch-formodningen* siger, at den nedre grænse for en pivotregels kompleksitet er $O(n)$, men dette blev modbevist i 2010. I [Kalai and Kleitman, 1992] er det vist, at der findes pivotregler med kompleksiteten $O(n^{1+ln(n)})$, hvilket er det bedste resultat fundet indtil videre, men ingen pivotregler med denne kompleksitet er fundet. Det er dog gældende for et tilfældigt lineært programmeringproblem, at der med meget høj sandsynlighed skulle fortages mellem $2m$ og $3m$ pivoteringer, og højest m^2, hvor m er antal betingelser for simplexalgoritmen. Altså er algortimen normalt hurtig nok, men det ville være bedre, hvis man kunne finde en anden algoritme, der har en polynomiel kompleksitet. [Newman, 2016] og [Matoušek and Gärtner, 2007].

5 | Ellipsoidealgoritmen

Simplexalgoritmens kompleksitet er et stort problem for algoritmen fra et teoretisk perspektiv, så andre algoritmer med lavere kompleksitet havde været ønsket i lang tid. Lineære programmeringsproblemer var set som NP-problemer indtil *ellipsoidemetoden* blev offentliggjort i 1979. Denne algoritme var den første algoritme, som løste lineære optimeringsproblemer i polynomiel tid. Denne algoritme virker fundamentalt forskelligt fra simplexalgoritmen, og derfor skal nye begreber introduceres.

5.1 Konvekse mængder

Først defineres, hvad det vil sige, at en mængde er *konveks*. Denne sektion tager udgangspunkt i [UIO, 20/01/2014].

Definition 5.1.1. *En mængde K er konveks, hvis $\lambda\overrightarrow{x} + (1-\lambda)\overrightarrow{y} \in K$, for alle $\lambda \in [0,1]$ og alle $\overrightarrow{x}, \overrightarrow{y} \in K$.*

En anden måde at formulere definitionen på er: Hvis du har to elementer $\overrightarrow{x}, \overrightarrow{y}$ i K, så kan du tegne et linjestykke mellem $\overrightarrow{x}$ og $\overrightarrow{y}$, sådan at linjestykket forbliver i K. Et eksempel på en konveks mængde kunne være alle de reelle tal, da summen af to reelle tal altid giver et reelt tal. Ud fra konvekse mængder kan et *konvekst problem* opstilles. Det defineres på følgende måde:

Definition 5.1.2. *Et problem siges at være et konvekst problem, hvis det er på formen*

$$\begin{aligned} &\textit{Maksimér: } z = \overrightarrow{c} \cdot \overrightarrow{x} \\ &\textit{hvor: } \qquad \overrightarrow{x} \in K, \end{aligned} \tag{5.1}$$

hvor K er en konveks mængde.

Enhver mulig mængde P, som et lineært programmeringsproblem afgrænser, er en *polytop*. Dette defineres nu.

Definition 5.1.3. *En n-polytop P er en mængde af punkter $\overrightarrow{x} \in \mathbb{R}^n$, der opfylder:*

$$P = \{\overrightarrow{x} \in \mathbb{R}^n | A \cdot \overrightarrow{x} \leq \overrightarrow{b}\}, \tag{5.2}$$

hvor $A \in \mathbb{R}^{m \times n}$ og $\overrightarrow{b} \in \mathbb{R}^m$.

Nu vises at en n-polytop er en konveks mængde, da dette beviser, at ethvert lineært programmerings problem kan beskrives som et konvekst optimeringsproblem.

Sætning 5.1.4. *Enhver n-polytop er en konveks mængde.*

Bevis. Lad en n-polytop P være givet ved $P = \{\vec{x} \in \mathbb{R}^n | A \cdot \vec{x} \leq \vec{b}\}$ og lad $\vec{x}_1$ og $\vec{x}_2$ være punkter i P og $\lambda \in [0,1]$. Da gælder følgende ud fra Definition 5.1.1 af konvekse mængder at:

$$A((1-\lambda) \cdot \vec{x}_1 + \lambda \cdot \vec{x}_2) = (1-\lambda) \cdot A \cdot \vec{x}_1 + \lambda \cdot A \cdot \vec{x}_2 \leq (1-\lambda) \cdot \vec{b} + \lambda \cdot \vec{b} = \vec{b}. \tag{5.3}$$

Den sidste ulighed gælder ud fra definitionen af en n-polytop. Dermed er det vist at $(1-\lambda) \cdot \vec{x}_1 + \lambda \cdot \vec{x}_2 \in P$. ■

Der indføres også det, der kaldes et *separationsorakel*, der anvendes i ellipsoidemetoden. For at beskrive separationsorakler, så beskrives *separerende hyperplaner* nu. Enhver hyperplan deler $\mathbb{R}^n$ op i to dele kaldet *halvrum*. Hvis en hyperplan er givet ved $\vec{a} \cdot \vec{x} = b$, så er de to halvrum H^+ og H^- givet ved $H^+ = \{\vec{x} \in \mathbb{R}^n | \vec{a} \cdot \vec{x} \geq b\}$ og $H^- : \{\vec{x} \in \mathbb{R}^n | \vec{a} \cdot \vec{x} \leq b\}$.

Definition 5.1.5. *Lad delmængderne $S, T \subset \mathbb{R}^n$ og H være en hyperplan i $\mathbb{R}^n$. H separerer S og T, hvis $S \subset H^+$ og $T \subset H^-$. Hvis H separerer to mængder, så kaldes H en separerende hyperplan. Hvis et eller flere punkter i S ligger på H og resten af S ligger i samme halvrum dannet af H, så siges H at være en støttende hyperplan.*

Separerende hyperplaner kan anvendes til at formulere følgende sætning, med henblik på at definere et separationsorakel.

Sætning 5.1.6. *Lad K være en lukket ikke-tom konveks mængde i $\mathbb{R}^n$ og $\vec{x}$ være et punkt i $\mathbb{R}^n$. Hvis $\vec{x}^* \notin K$, så findes en separerende hyperplan, der separerer $\vec{x}^*$ og K.*

Bevis. Lad $\vec{x}^*$ være et punkt, hvorom det gælder, at $\vec{x}^* \notin K$, og $\vec{x}_0$ være det punkt på randen, som er nærmest $\vec{x}^*$. Beviset for at et sådan punkt eksisterer udføres ikke i dette projekt. Lad desuden $\vec{p} = \vec{x}_0 - \vec{x}^*$ og $b = \vec{p} \cdot \vec{x}_0$.

Først vises det at $\vec{p} \cdot \vec{x}^* < b$. Her er $\vec{p} \cdot \vec{x}_0 - \vec{p} \cdot \vec{x}^* = \vec{p} \cdot (\vec{x}_0 - \vec{x}^*) = \vec{p} \cdot \vec{p}$. Da $\vec{x}^* \neq \vec{x}_0$, så er $\vec{p} \cdot \vec{p} > 0$. Nu kan det findes at

$$\begin{aligned} \vec{p} \cdot \vec{x}_0 - \vec{p} \cdot \vec{x}^* &> 0 \\ b = \vec{p} \cdot \vec{x}_0 &> \vec{p} \cdot \vec{x}^*. \end{aligned} \tag{5.4}$$

Altså er $\vec{p} \cdot \vec{x}^* < b$. Det ønskes nu vist at for alle $\vec{x} \in K$, så er $\vec{p} \cdot \vec{x} \geq b$. Dette bevises med modstrid, så antag at $\vec{p} \cdot \vec{x} < b$. Lad $\vec{x}_\lambda = (1-\lambda)\vec{x}_0 + \lambda\vec{x}$ for alle $\lambda \in [0,1]$. Ud fra Definition 5.1.1, så er $\vec{x}_\lambda \in K$ for $\lambda \in [0,1]$.

Nu ønskes det at vise at $d(\vec{x}^*, \vec{x}_\lambda) < d(\vec{x}^*, \vec{x}_0)$ Her betegner $d(\vec{a}, \vec{b})$ afstanden mellem punkterne $\vec{a}$ og $\vec{b}$. Dette vil modstride antagelsen om, at $\vec{x}_0$ er det punkt nærmest $\vec{x}^*$. Først vises at:

$$\begin{aligned} \vec{x}_\lambda - \vec{x}^* &= (1-\lambda)\vec{x}_0 + \lambda\vec{x} - \vec{x}^* \\ &= (\vec{x}_0 - \vec{x}^*) + \lambda(\vec{x} - \vec{x}_0) \\ &= \vec{p} + \lambda(\vec{x} - \vec{x}_0) \end{aligned} \tag{5.5}$$

Nu vises at $d(\vec{x}^*, \vec{x}_\lambda) < d(\vec{x}^*, \vec{x}_0)$. Dette er det samme som uligheden $|\vec{x}^* - \vec{x}_\lambda|^2 < |\vec{x}^* - \vec{x}_0|^2$.

$$\begin{aligned} |\vec{x}^* - \vec{x}_\lambda|^2 &= (\vec{x}^* - \vec{x}_\lambda)\cdot(\vec{x}^* - \vec{x}_\lambda) \\ &= (\vec{p} + \lambda(\vec{x} - \vec{x}_0))\cdot(\vec{p} + \lambda(\vec{x} - \vec{x}_0)) \\ &= \vec{p}\cdot\vec{p} + 2\lambda\vec{p}\cdot(\vec{x} - \vec{x}_0) + \lambda^2|\vec{x} - \vec{x}_0|^2 \\ &= |\vec{x}_0 - \vec{x}^*|^2 + \lambda g(\lambda) \end{aligned} \tag{5.6}$$

Her er $g(\lambda) = 2(\vec{p}\cdot\vec{x} - \vec{p}\cdot\vec{x}_0) + \lambda|\vec{x} - \vec{x}_0|^2$, og da $\vec{p}\cdot\vec{x}_0 > \vec{p}\cdot\vec{x}$ per antagelse, så er $g(\lambda) = 2\xi + \lambda|\vec{x} - \vec{x}_0|^2$, hvor $\xi < 0$. Herved findes det at $\lim_{\lambda\to 0} g(\lambda) = 2\xi$. Da $\xi < 0$, så findes der værdier af λ, hvor $|\vec{x}^* - x_\lambda|^2 < |\vec{x}^* - \vec{x}_0|^2$, og derved $d(\vec{x}^*, \vec{x}_\lambda) < d(\vec{x}^*, \vec{x}_0)$, hvilket er en modstrid, og derved er $\vec{p}\cdot\vec{x} \geq b$ for alle $\vec{x} \in K$.

Da $\vec{p}\cdot\vec{x} \geq b$ og $\vec{p}\cdot\vec{x}^* < b$, så er $\vec{p}\cdot\vec{x} = b$ en støttende hyperplan, der separerer K og $\vec{x}^*$. En hyperplan der går gennem x', og danner et halvrum der inkluderer K, er nu givet ved $\vec{p}\cdot\vec{x} = c$, hvor $c = \vec{p}\cdot\vec{x}^*$. Da der kan vælges et tal mellem b og c, så findes der også en separerende hyperplan, som hverken inkluderer punkter i K eller $\vec{x}^*$, og Sætning 5.1.6 er bevist. [Walker, 2017]. ■

Bevis 5.1.6 anvendes til at definere det, der hedder et separationsorakel.

Definition 5.1.7. *Et separationsorakel for en konveks mængde K, er en procedure, der som input tager et punkt $\vec{p}$ og returnerer om $\vec{p} \in K$ eller returnerer en hyperplan, der separerer $\vec{p}$ og hele K.*

Man kan anvende et separationsorakel på et lineært programmeringsproblem. Hvis et lineært programmeringsproblem har betingelserne $\vec{a}_1\cdot\vec{x} \leq b_1, \vec{a}_2\cdot\vec{x} \leq b_2, \dots, \vec{a}_m\cdot\vec{x} \leq b_m$, så indsættes et punkt $\vec{p}$ i betingelserne en efter en. Hvis $\vec{p}$ overholder alle betingelser, så er $\vec{p} \in K$, og hvis $\vec{p}$ ikke overholder en betingelse $\vec{a}_i\cdot\vec{p} \leq b_i$, så returneres hyperplanen $\vec{a}_i\cdot\vec{x} = b_i$, hvilket er en støttende hyperplan til K, der separerer K og $\vec{p}$. At finde et separationsorakel forgår med $O(n\cdot m)$ kompleksitet, da det blot kræver et tjek per betingelse.

5.2 Konvekse og konkave funktioner

Konvekse og konkave funktioner er anvendelige i forhold til at bestemme, hvorvidt et område er konvekst eller konkavt, derfor defineres disse begreber.

Definition 5.2.1. *Lad en konveks mængde $C \subseteq \mathbb{R}^n$ være givet. En funktion $f : C \to \mathbb{R}$ siges at være konveks, hvis der for alle $\vec{x}, \vec{y} \in C$ og for alle $\lambda \in [0;1]$ gælder:*

$$f(\lambda \cdot \vec{x} + (1-\lambda) \cdot \vec{y}) \leq \lambda \cdot f(\vec{x}) + (1-\lambda) \cdot f(\vec{y}).$$

Ligeledes siges en funktion at være konkav, hvis der for alle $\vec{x}, \vec{y} \in C$ og for alle $\lambda \in [0;1]$, gælder:

$$f(\lambda \cdot \vec{x} + (1-\lambda) \cdot \vec{y}) \geq \lambda \cdot f(\vec{x}) + (1-\lambda) \cdot f(\vec{y}).$$

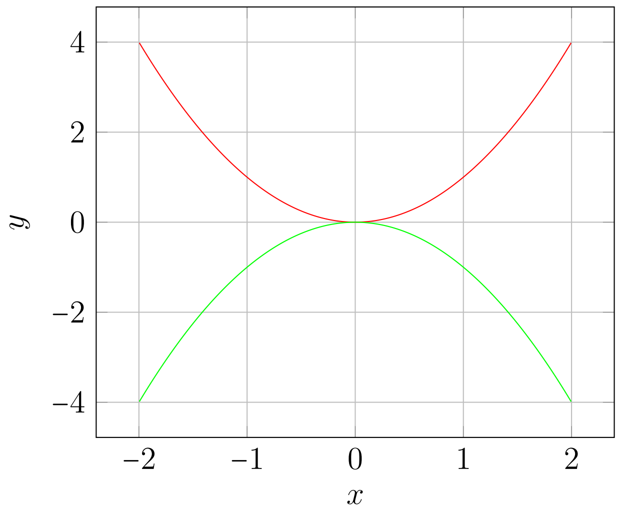
Som et eksempel gennemgås et bevis for at funktionen $f(x) = x^2$ er konveks og ikke konkav.

Eksempel 5.2.2. Lad funktionen $f(x) = x^2$ for alle $x \in \mathbb{R}$ være givet. Denne funktion er konveks, men ikke konkav. For at bevise det, antages det at funktionen er konveks, og derefter bevises det at uanset hvilke værdier x og y tager, vil uligheden være opfyldt.

$$\begin{aligned}
&\text{Det skal vises at:}\\
&(\lambda \cdot x + (1-\lambda) \cdot y)^2 \leq \lambda \cdot x^2 + (1-\lambda) \cdot y^2 \qquad (5.7)\\
&\text{Derfor regnes på:}\\
&(\lambda \cdot x + (1-\lambda) \cdot y)^2 - \lambda \cdot x^2 - (1-\lambda) \cdot y^2\\
=&\lambda^2 \cdot x^2 + (1-\lambda)^2 \cdot y^2 + 2 \cdot \lambda \cdot (1-\lambda) \cdot x \cdot y - \lambda \cdot x^2 - (1-\lambda) \cdot y^2\\
=&\lambda^2 \cdot x^2 + y^2 + \lambda^2 \cdot y^2 - 2\lambda \cdot y^2 + 2 \cdot \lambda \cdot x \cdot y - 2 \cdot \lambda^2 \cdot x \cdot y - \lambda \cdot x^2 - y^2 + \lambda \cdot y^2\\
=&\lambda^2 \cdot x^2 + \lambda^2 \cdot y^2 - \lambda \cdot y^2 + 2 \cdot \lambda \cdot x \cdot y - 2 \cdot \lambda^2 \cdot x \cdot y - \lambda \cdot x^2\\
=&(\lambda - 1) \cdot \lambda \cdot x^2 + (\lambda - 1) \cdot \lambda \cdot y^2 - 2 \cdot \lambda^2 \cdot x \cdot y + 2 \cdot \lambda \cdot x \cdot y\\
=&(\lambda - 1) \cdot \lambda \cdot x^2 + (\lambda - 1) \cdot \lambda \cdot y^2 + 2 \cdot (1-\lambda) \cdot \lambda \cdot x \cdot y\\
=&(\lambda - 1) \cdot \lambda \cdot x^2 + (\lambda - 1) \cdot \lambda \cdot y^2 - (\lambda - 1) \cdot \lambda \cdot 2 \cdot x \cdot y\\
=&(\lambda - 1) \cdot \lambda \cdot (x^2 + y^2 - 2 \cdot x \cdot y)\\
=&(\lambda - 1) \cdot \lambda \cdot (x - y)^2 \leq 0
\end{aligned}$$

Da λ er mellem 0 og 1 får $(\lambda - 1)$ negativt fortegn, og siden differensen mellem x og y kvadreres, vil dette give et positivt tal uanset valget af x og y. Dermed ganges et negativt tal med et tal mellem 0 og 1 og et positivt tal, hvilket må medføre at venstresiden samlet set giver et negativt tal. Dermed er uligheden for konveksitet opfyldt, men vendes uligheden for at undersøge om funktionen er konkav er uligheden kun opfyldt for $\lambda = 0 \lor \lambda = 1$, da disse giver værdien 0. Siden det kræves at uligheden skal være sand for alle λ er funktionen derfor ikke konkav.

Rent grafisk har konkave og konvekse funktioner også en betydning. Betragt grafen for henholdsvis $f(x) = x^2$ og $g(x) = -x^2$ i Figur 5.1.



Figur 5.1: Konvekse og konkave funktioner

Herudfra ses, at $f(x)$ er konveks og $g(x)$ er konkav. Dette observeres ud fra hvilken vej funktionen krummer, hvis den krummer opad er funktionen konveks, hvis den krummer nedad er funktionen konkav. Til sidst skal det nævnes, at en funktion af typen $f(\overrightarrow{x}) = \sum_{i=1}^{n} a_i \cdot x_i$ både er konveks og konkav, idet der gælder følgende:

$$\begin{aligned} f(\lambda \cdot \overrightarrow{x} + (1-\lambda) \cdot \overrightarrow{y}) &= f(\lambda \cdot \overrightarrow{x}) + f((1-\lambda) \cdot \overrightarrow{y}) \\ &= \lambda f(\overrightarrow{x}) + (1-\lambda) f(\overrightarrow{y}). \end{aligned}$$

Da f er lineær, må f være både konkav og konveks.

5.3 Ellipsoider

Eftersom ellipsoidemetoden bygger på *ellipsoider*, som er n-dimensionale ellipser, så skal de defineres.

Definition 5.3.1. *En ellipsoide i $\mathbb{R}^n$ er mængden af punkter der opfylder:*

$$E(\overrightarrow{a}, A) = \{\overrightarrow{x} \in \mathbb{R}^n | (\overrightarrow{x} - \overrightarrow{a})^T A (\overrightarrow{x} - \overrightarrow{a}) \leq 1\},$$

hvor A er en positiv definit matrix og $\overrightarrow{a}$ er ellipsoidens midtpunkt.

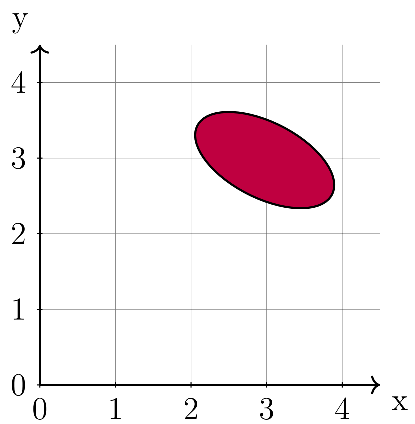
En hyperkugle er et specialtilfælde af en ellipsoide, hvor $A = R \cdot I_n$, og R er hyperkuglens radius. Hyperkugler benævnes som $\Theta(\overrightarrow{a}, R)$. Her er $\overrightarrow{a}$ kuglens midtpunkt, og R er kuglens radius.

For at få en bedre forståelse for ellipsoider gennemgås et eksempel på en ellipsoide.

Eksempel 5.3.2. Betragt følgende ellipsoide:

$$E(\overrightarrow{a}, A) = [(x-3)\ (y-3)] \cdot \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \cdot \begin{bmatrix} x-3 \\ y-3 \end{bmatrix} \leq 1. \tag{5.8}$$

Dette kan reduceres til $3 \cdot x^2 + 4 \cdot x \cdot y + 6 \cdot y^2 - 30 \cdot x - 48 \cdot y + 117 \leq 1$. Dette beskriver en ellipse i planen, som kan ses på Figur 5.2.



Figur 5.2: Ellipsiode i $\mathbb{R}^2$.

Sætning 5.3.3. *Ellipsoider er konvekse.*

Bevis. Da A er positiv definit, så kan den skrives som $A = B^T B$. Derved findes det, at

$$(\overrightarrow{x} - \overrightarrow{a})^T B^T B(\overrightarrow{x} - \overrightarrow{a}) = (B(\overrightarrow{x} - \overrightarrow{a}))^T (B(\overrightarrow{x} - \overrightarrow{a})) = |B(\overrightarrow{x} - \overrightarrow{a})|^2 \leq 1.$$

Dette kan omskrives til

$$\begin{aligned} |B(\overrightarrow{x} - \overrightarrow{a})|^2 &\leq 1 \\ \sqrt{|B(\overrightarrow{x} - \overrightarrow{a})|^2} &\leq \sqrt{1} \\ |B(\overrightarrow{x} - \overrightarrow{a})| &\leq 1. \end{aligned}$$

Lad nu dette gælde om punkterne $\overrightarrow{x}$ og $\overrightarrow{y}$, så skal det også gælde om punktet $\lambda\overrightarrow{x}+(1-\lambda)\overrightarrow{y}$, for alle $\lambda \in [0;1]$, for at ellipsoiden er konveks. Ved brug af trekantsuligheden findes det nu at:

$$\begin{aligned} |B(\lambda\overrightarrow{x} + (1-\lambda)\overrightarrow{y} - \overrightarrow{a})| &= \\ |B(\lambda(\overrightarrow{x} - \overrightarrow{a}) + (1-\lambda)(\overrightarrow{y} - \overrightarrow{a}))| &\leq \\ \lambda|B(\overrightarrow{x} - \overrightarrow{a})| + (1-\lambda)|B(\overrightarrow{y} - \overrightarrow{a})| &\leq \\ \lambda \cdot 1 + (1-\lambda) \cdot 1 &= 1. \end{aligned}$$

Derved er Sætning 5.3.3 bevist. ∎

5.4 Algoritmen

Denne sektion tager udgangspunkt i [MIT, 18/03/2016]. Ud fra et optimeringsproblem på standardform kan der dannes en ny mulig mængde givet på følgende måde:

$$\begin{aligned} \overrightarrow{c} \cdot \overrightarrow{x} &\geq c_0 \\ A \cdot \overrightarrow{x} &\leq \overrightarrow{b} \\ \overrightarrow{x} &\geq \overrightarrow{0}. \end{aligned} \tag{5.9}$$

Her er c_0 et gæt på den optimale værdi af objektfunktionen. Læg mærke til $\overrightarrow{c} \cdot \overrightarrow{x} \geq c_0$ blot skaber et halvrum. Nu kan det tjekkes om det nye område er muligt. Hvis det er muligt, så kan det findes at $z' \geq c_0$, og ellers er $z' \leq c_0$. Herved kan en søgningsalgoritme, som eksempelvis binær søgning, anvendes til at finde den optimale værdi, ved at danne nye gæt, som indskrænker halvrummet, som indeholder den optimale værdi. Der eksisterer sådanne algoritmer, som har polynomiel kompleksitet. Det eneste, der er påkrævet, er at finde en algoritme til at tjekke om et optimeringsproblem er muligt. Dette beskrives nu.

Der tages udgangspunkt i det konvekse optimeringsproblem beskrevet i Afsnit 5.1. Algoritmen beskrives som følgende:

- Start med en hyperkugle $\Theta(\overrightarrow{0}, R) = E_0(\overrightarrow{0}, R \cdot I_n)$, der med garanti indeholder det mulige område P.
- Fra $k = 0$ til m lad $E_k(\overrightarrow{a}_k, A_k)$ være den nuværende ellipsoide. Anvend et separationsorakel til at undersøge om $\overrightarrow{a}_k$ er i P. Hvis $a_k \in P$ så terminerer algoritmen, da problemet så er muligt.
- Hvis $a_k \notin P$, dan da en ny ellipsoide E_{k+1}, der har mindre volumen end E_k.

Algoritmen er afhængig af, at $P \neq \varnothing$, og at P er fulddimensionel i $\mathbb{R}^n$. Det vil sige, at P er en n-polytop. Hvis $P = \varnothing$, så terminerer algoritmen stadig, da der er en øvre grænse på mængden af iterationer. Dette opskrives i Sætning 5.4.3. Det skal nu vises, hvordan den næste ellipsoide i algoritmen kan dannes.

Sætning 5.4.1. *Hvis* $E = E(\overrightarrow{0}, I_n)$, *hvor* I_n *er identitetsmatricen, og* $\overrightarrow{c} = -\overrightarrow{e}_1$, *hvor* $\overrightarrow{c}$ *er den vektor, der står vinkelret på den separerende hyperplan i retning væk fra den konvekse mængde, så kan den nye ellipsoide skrives som følgende, uden tab af generalitet:*

$$E^* = \{\overrightarrow{x} \in \mathbb{R}^n | \left(\frac{n+1}{n}\right)^2 \cdot \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2} \cdot \sum_{j=2}^{n} x_j^2 \le 1\}. \tag{5.10}$$

Bevis. For at vise, at dette kan bruges som den nye ellipsoide, skal der gælde to ting.

1. Hvis $\overrightarrow{x} \in E$, og $x_1 \ge 0$, så er $\overrightarrow{x} \in E^*$.
2. Volumen af E^* skal være mindre end volumen af E.

Først vises punkt nummer 1. Vi indsætter $\overrightarrow{x}$ fra E i udtrykket for E^*, og ser at uligheden gælder.

$$\frac{n^2+1+2n}{n^2} \cdot \left(x_1^2 + \frac{1}{n^2+1+2n} - \frac{2x_1}{n+1}\right) + \frac{n^2-1}{n^2} \cdot \sum_{j=2}^{n} x_j^2 =$$
$$\frac{n^2+1+2n}{n^2} \cdot x_1^2 + \frac{1}{n^2} - \frac{2 \cdot x_1 \cdot (n+1)}{n^2} + \frac{n^2-1}{n^2} \cdot \sum_{j=2}^{n} x_j^2 =$$
$$\left(\frac{n^2-1}{n^2} + \frac{2n+2}{n^2}\right) \cdot x_1^2 + \frac{1}{n^2} - \frac{2 \cdot x_1 \cdot (n+1)}{n^2} + \frac{n^2-1}{n^2} \cdot \sum_{j=2}^{n} x_j^2 =$$
$$\left(\frac{2n+2}{n^2}\right) \cdot x_1^2 + \frac{1}{n^2} - \frac{2 \cdot x_1 \cdot (n+1)}{n^2} + \frac{n^2-1}{n^2} \cdot \sum_{j=1}^{n} x_j^2 =$$
$$\left(\frac{(2n+2) \cdot (x_1^2 - x_1)}{n^2}\right) + \frac{1}{n^2} + \frac{n^2-1}{n^2} \cdot \sum_{j=1}^{n} x_j^2.$$

Da $\overrightarrow{x}$ var i den oprindelige ellipsoide E vil det sige, at $\sum_{j=1}^{n} x_j^2 \le 1$. Da $x_1^2 - x_1 \le 0$, når $0 \le x_1 \le 1$, medfører det at $\frac{(2n+2)\cdot(x_1^2 - x_1)}{n^2} \le 0$. Ud fra dette følger det at:

$$\left(\frac{(2n+2) \cdot (x_1^2 - x_1)}{n^2}\right) + \frac{1}{n^2} + \frac{n^2-1}{n^2} \cdot \sum_{j=1}^{n} x_j^2 \le 1. \tag{5.11}$$

Hermed er det vist at $x \in E^*$. Nu skal det vises, at volumen af E^* er mindre end volumen af E.
For at bevise dette, noteres det at volumen af en ellipsoide er proportionel med produktet af ellipsoidens akselængder. Derudover er koefficienterne $\frac{1}{(\frac{n+1}{n})}$ og $\frac{1}{\sqrt{\frac{n^2-1}{n^2}}}$ akselængderne for

E^*. Dette vises ikke. Nu kan vi opskrive forholdet mellem $\mathrm{Vol}(E^*)$ og $\mathrm{Vol}(E)$.

$$\frac{Vol(E^*)}{Vol(E)} = \frac{n}{n+1} \cdot \left(\sqrt{\frac{n^2}{n^2-1}}\right)^{n-1} = \left(\frac{-1}{n+1}+1\right)\left(\frac{1}{n^2-1}+1\right)^{\frac{n-1}{2}} \leq e^{\frac{-1}{n+1}} \cdot e^{\frac{n-1}{2\cdot(n^2-1)}} = e^{\frac{-1}{2\cdot(n+1)}} \tag{5.12}$$

Uligheden gælder da $x+1 \leq e^x$. Dette bevirker, at (5.10) er en kandidat til den næste ellipsoide. ■

Lad os nu undersøge den generelle situation, hvor ellipsoiden er givet ved $E(\overrightarrow{a}, A)$. Dette kan reduceres til det specielle tilfælde beskrevet i Sætning 5.4.1, hvilket gøres ved at observere følgende. Da A er en positiv definit matrix, er $A = B^T B$ gældende for en invertibel matrix B. Invertible lineære transformationer ændrer ikke forholdet mellem volumen af objekter. Nu bestemmes den lineære transformation $T(\overrightarrow{x})$, der transformerer $E(\overrightarrow{a}, A)$ til $E(\overrightarrow{0}, I)$.

Sætning 5.4.2. *Lad* $T(\overrightarrow{x}) = B \cdot (\overrightarrow{x} - \overrightarrow{a})$. *Da er* $T(E(\overrightarrow{a}, A)) = E(\overrightarrow{0}, I)$.

Bevis. I Sætning (5.3.3) var det vist at

$$E(\overrightarrow{a}, A) = \{\overrightarrow{x} \in \mathbb{R}^n \| B(\overrightarrow{x} - \overrightarrow{a})| \leq 1\}.$$

Derved findes det at

$$T(E(\overrightarrow{a}, A)) = \{T(\overrightarrow{x}) \in \mathbb{R}^n \| B(\overrightarrow{x} - \overrightarrow{a})| \leq 1\}.$$

Et punkt $\overrightarrow{y}$ defineres som $\overrightarrow{y} = T(\overrightarrow{x})$. Derved er

$$T(E(\overrightarrow{a}, A)) = \{\overrightarrow{y} \in \mathbb{R}^n \| \overrightarrow{y}| \leq 1\},$$

hvilket her ses at være enhedskuglen $\Theta(\overrightarrow{0}, 1)$. ■

Nu bestemmes en begrænsning på mængden af iterationer. Dette gøres ved at betragte en hyperkugle givet ved $\Theta(\overrightarrow{q}, r)$, hvor $\overrightarrow{q}$ er et punkt i P, som ikke er på randen. Hvis $P \neq \varnothing$, så er $\Theta(\overrightarrow{q}, r) \subseteq P$.

Sætning 5.4.3. *Hvis* $N > 2n(n+1) \cdot \ln(\frac{R}{r})$, *hvor* R *er* E_0*'s radius, så er* $\mathrm{Vol}(E_N) \leq \mathrm{Vol}(\Theta(\overrightarrow{q}, r))$ *gældende efter* N *iterationer.*

Bevis. Beviset tager udgangspunkt i [O'Donnell, 2011]. Volumen af den n'te ellipsoide findes ud fra Sætning 5.4.1. Dette gøres på følgende måde:

$$\mathrm{Vol}(E_N) \leq \mathrm{Vol}(E_0) \cdot e^{\frac{-N}{2(n+1)}}. \tag{5.13}$$

Dette gælder fordi:

$$\frac{\mathrm{Vol}(E_N)}{\mathrm{Vol}(E_{N-1})} \leq e^{\frac{-1}{2(n+1)}}. \tag{5.14}$$

$$\mathrm{Vol}(E_N) \leq e^{\frac{-1}{2(n+1)}} \cdot \mathrm{Vol}(E_{N-1}). \tag{5.15}$$

Ved brug af Sætning 5.4.1 kan vi opnå et udtryk for $\mathrm{Vol}(E_{N-1})$:

$$\mathrm{Vol}(E_{N-1}) \leq e^{\frac{-1}{2(n+1)}} \cdot \mathrm{Vol}(E_{N-2}). \tag{5.16}$$

Nu indsættes dette udtryk i Ulighed (5.15) og derved fås.

$$\mathrm{Vol}(E_N) \leq e^{\frac{-2}{2(n+1)}} \cdot \mathrm{Vol}(E_{N-2}). \tag{5.17}$$

Denne proces fortsættes indtil Ulighed (5.13) gælder. Nu isoleres der for $\mathrm{Vol}(E_N)$.

$$\mathrm{Vol}(E_N) \leq \mathrm{Vol}(E_0) \cdot e^{\frac{-N}{2\cdot(n+1)}} < \mathrm{Vol}(\Theta(0,R)) \cdot e^{-n\cdot\ln(\frac{R}{r})}. \tag{5.18}$$

Dette kan omformes til:

$$\mathrm{Vol}(E_N) < R^n \cdot \mathrm{Vol}(\Theta(\overrightarrow{0},1)) \cdot \frac{r^n}{R^n} = \tag{5.19}$$

$$\mathrm{Vol}\big(\Theta(\overrightarrow{0},1)\big) \cdot r^n = \mathrm{Vol}\big(\Theta(\overrightarrow{0},r)\big) = \mathrm{Vol}\big(\Theta(\overrightarrow{q},r)\big). \tag{5.20}$$

■

Korollar 5.4.4. *$P = \varnothing$ efter $2n(n+1) \cdot ln\left(\frac{R}{r}\right)$ iterationer.*

Bevis. Da $B(\overrightarrow{q},r) \subseteq P$ og $P \subseteq E_k$ var antaget, og det er vist at $\mathrm{Vol}(E_N) < \mathrm{Vol}(\Theta(\overrightarrow{q},r))$, så er der en modstrid efter $2n(n+1)\cdot\ln\left(\frac{R}{r}\right)$ iterationer. Derved må $P = \varnothing$. ■

5.4.1 Algoritmeoversigt

Nu kan en konkret algoritme beskrives. Lad Q være et muligt lineært programmeringsproblem på standardform med objektfunktion $z = \overrightarrow{c}\cdot\overrightarrow{x}$. Lad l og u være tal således tilføjelsen af betingelsen $\overrightarrow{c}\cdot\overrightarrow{x} \geq u$ gør Q ikke-muligt, mens efter tilføjelsen af betingelsen $\overrightarrow{c}\cdot\overrightarrow{x} \geq l$, så vil Q stadigt være muligt. Der er også påkrævet et $\varepsilon > 0$, hvilket er den acceptable afvigelse, som er den højest mulig afstand mellem algoritmens resultat og det lineær programmeringsproblems optimale objektfunktion. Derudover starter ellipsoidealgoritmen ved $j = 0$, og slutter når $j > N$, hvor $N = 2n(n+1)\cdot\ln\left(\frac{R}{r}\right)$. Nu forløber algoritmen som følgende:

1. Dan et område P som det mulige område, fundet ved at tilføje en betingelse $\overrightarrow{c}\cdot\overrightarrow{x} \geq c_0$, hvor $c_0 = \frac{l+u}{2}$, til Q.

2. Tjek om problemet nu er muligt via ellipsoidemetoden.
   (a) Dan den initierende ellipsoide givet ved $E_0(\overrightarrow{a}_0, R \cdot I_n)$, hvor $\overrightarrow{a}_0 = \overrightarrow{0}$. Denne ellipsoide indeholder P, hvis et stort nok R vælges.
   (b) Anvend separationsorakel: Hvis $\overrightarrow{a}_j \in P$, så returneres problemet som muligt, og gå til trin 3. Hvis ikke, så bestem en hyperplan, der separerer $\overrightarrow{a}_j$ og hele P.
   (c) Transformér $E_j(a_j, A_j)$ til enhedskuglen ved brug af transformationen $T(\overrightarrow{x}) = B_j \cdot (\overrightarrow{x} - \overrightarrow{a}_j)$. Samme transformation bruges på hyperplanen.
   (d) Brug formlen for ellipsoiden beskrevet i Ligning 5.10 til at bestemme E^*
   (e) Transformér tilbage igen ved brug af den inverse lineære transformation. Nu er $E_{j+1}(a_{j+1}, A_{j+1})$ bestemt.
   (f) Forøg værdien af j med 1 og gentag fra punkt (b). Hvis $j > N$, så returnér problemet som ikke-muligt og gå til trin 3.
3. Hvis problemet er muligt, så lad $l = c_0$, og hvis problemet ikke er muligt, så lad $u = c_0$.
4. Gentag algoritmen indtil $|l - u| < \varepsilon$, hvorefter $z' = \frac{l+u}{2}$ returneres.

For at bestemme l og u kan trin (a) til (e) bruges på Q. Derved bestemmes to punkter, et i Q, og et udenfor, som kan bruges til at bestemme l og u. I algoritmen skal R og r bestemmes. Her vælges bare et yderst stort R og en yderst lille r.

5.4.2 Kompleksitet

Ellipsoidealgoritmen tager udgangspunkt i binær søgning, som har en logaritmisk kompleksitet. Derfor er algoritmen afhængig af metoden til at tjekke en problem er mulig. Dette er her ellipsoidemetoden anvendes. At beregne kompleksiteten af dette er ud over dette projekts rammer, men det er stadigt centralt for algoritmen. Ellipsoidemetoden har en worstcase-kompleksitet på $O(n^4 \log(\frac{1}{\varepsilon}))$, hvilket er polynomielt. Dette er den største grund til at bruge ellipsoidealgoritmen fremfor simplexalgoritmen. Dog er ellipsoidealgoritmen ikke brugt i praksis, da der er mere effektive polynomielle algoritmer, samt algoritmer som simplexalgoritmen, som ikke er polynomiel teoretisk set, men er effektiv i praksis. Derved er ellipsoidealgoritmen eksklusivt interessant ved diskussioner i teoretiske sammenhæng. [Bubeck, 2013]

6 Anvendelse

I dette kapitel opstilles et lineært programmeringseksempel, hvor simplexalgoritmen skal anvendes til at løse problemet. Eksemplet vil besidde så mange variable og begrænsninger, at det i praksis ikke vil være muligt at løse problemet i hånden indenfor en rimelig tidsperiode. Derfor har vi programmeret simplexalgoritmen i Python3 til at løse problemer af denne type. Vi har valgt ikke at programmere ellipsoidealgoritmen, da det var for ressource- og tidskrævende for projektets rammer. Den vil dog stadig blive diskuteret, og sammenlignet med simplexalgoritmen. Python3 koden findes i Appendix A, hvor Appendix A.1 indeholder frameworket, der definerer matrix- og vektorregning, og selve simplexalgoritmen findes i Appendix A.2.
Eksemplet er med 12 variable og lyder:

$$
\begin{aligned}
\text{Maksimer: } z = & -8x_1 + 3x_2 + 3x_3 - 3x_4 + 3x_5 - 4x_6 + 3x_7 + 3x_8 - 2x_9 + 6x_{10} + 3x_{11} - 6x_{12} \\
\text{Betinget af: } & 5x_1 + 3x_2 - 4x_3 - 7x_4 - 2x_6 - 9x_7 + 6x_9 + 9x_{10} + 3x_{11} + 4x_{12} \leq 3 \\
& -6x_1 + x_2 - x_3 + 6x_4 + 4x_5 - 7x_6 + 5x_7 - 9x_8 - 6x_9 + 3x_{10} - 8x_{11} + 9x_{12} \leq 2 \\
& -x_1 + 5x_2 + 9x_3 - 4x_4 + 3x_5 - 5x_6 - 4x_7 + 5x_8 - 3x_9 - 3x_{10} - 8x_{11} - 9x_{12} \leq 7 \\
& -6x_1 + 4x_2 - 8x_3 - 6x_4 + 9x_5 + 3x_6 + 7x_7 + 8x_8 + 4x_9 - 4x_{10} - 6x_{11} - 3x_{12} \leq 8 \\
& 7x_1 - x_2 - 4x_3 + 3x_4 + 9x_5 - 5x_6 - 6x_7 - x_8 - 2x_9 - 5x_{10} - 9x_{11} - 7x_{12} \leq 9 \\
& 7x_1 + x_2 + 6x_3 + x_4 + 4x_5 + x_6 - 8x_7 - 2x_8 - 4x_9 - 6x_{10} + 7x_{11} \leq 2 \\
& 9x_1 - 3x_2 + 5x_3 + 9x_4 - 5x_5 + 2x_6 + x_7 + 2x_8 + 9x_9 - 3x_{10} - 6x_{11} \leq 6 \\
& -2x_1 + 2x_2 - 8x_3 - 3x_4 - 9x_5 + 9x_6 + x_7 + 8x_8 + 9x_9 - 4x_{10} - 3x_{11} + 7x_{12} \leq 9 \\
& 8x_1 - 4x_2 - 8x_4 - 7x_5 + 6x_7 - 2x_8 - 4x_9 - 3x_{10} - 6x_{11} + 6x_{12} \leq 8 \\
& -8x_1 - 9x_2 + 2x_3 + 7x_4 + x_5 - 8x_6 - 2x_7 + x_8 + 5x_9 + \quad + 2x_{11} - x_{12} \leq 5 \\
& x_1 + x_2 - 2x_3 + 6x_4 - 4x_5 + 5x_6 + x_7 + 2x_8 + 4x_9 - x_{10} + 8x_{11} - 3x_{12} \leq 3 \\
& 3x_1 + 6x_2 - 4x_3 - 5x_4 - 9x_5 + 3x_6 + 4x_7 + 4x_8 - 5x_9 + 9x_{10} - 2x_{11} + 6x_{12} \leq 7 \\
& x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0
\end{aligned}
\tag{6.1}
$$

Koden anvendes på problemet, hvor den som input tager: Objektvektoren $\overrightarrow{c}$, betingelsesvektoren $\overrightarrow{b}$ og koefficientmatricen A, hvor den opstiller dem som en tableaumatrix. Koden giver så som output, den endelige tableaumatrix og løsningen. bemærk at tableaumatricen ikke indeholder den andensidste søjle, da det ikke var nødvendigt for problemet. Den viser yderligere hvilke indgange, der er blevet pivoteret. Koden var 0.004 sekunder om at løse problemet på en computer med en 1.5 GHz processor. Det kalder man effektivt!

$$
\left[\begin{array}{cccccccccccc|cccccccccccc|c}
1.8 & 5.4 & 0 & 4.8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1.9 & 0.2 & 0.6 & 0.3 & 0.2 & 0 & 0.3 & 0.2 & 0 & 0 & 0 & 0.8 & 0.2 & 10.8 \\
-0.3 & 3 & 0 & 3.9 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0.1 & 0.1 & 0.4 & 0.2 & 0.1 & 0 & 0.1 & 0.1 & 0 & 0 & 0 & 0.6 & 0.1 & 6.1 \\
1.8 & 5.2 & 1 & 5.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.2 & 0.6 & 0.3 & 0.2 & 0 & 0.4 & 0.2 & 0 & 0 & 0 & 0.8 & 0.2 & 11.1 \\
1.1 & 4.3 & 0 & 3.8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1.2 & 0.2 & 0.5 & 0.2 & 0.2 & 0 & 0.3 & 0.1 & 0 & 0 & 0 & 0.6 & 0.1 & 8.7 \\
22 & 57 & 0 & 73 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.8 & 1.9 & 6.2 & 3.5 & 1.5 & 1 & 3.2 & 1.7 & 0 & 0 & 0 & 10.4 & 2.3 & 128.2 \\
3.2 & 3.1 & 0 & 3.2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2.8 & 0.1 & 0.3 & 0.1 & 0.2 & 0 & 0.4 & 0.2 & 0 & 0 & 0 & 0.4 & 0.2 & 7.5 \\
12.8 & 0 & 0 & 13.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 21 & 0.2 & 0.9 & -0.7 & 0.9 & 0 & 2.2 & 1.2 & 0 & 0 & 1 & -0.1 & 1.5 & 31 \\
3.2 & 66.6 & 0 & 63.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18.5 & 2.3 & 7.3 & 4.3 & 1.7 & 0 & 3.7 & 1.1 & 1 & 0 & 0 & 10.4 & 1.8 & 132.2 \\
19.2 & 36.6 & 0 & 24.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12.8 & 2 & 3.9 & 2.3 & 1.5 & 0 & 2.6 & 0.9 & 0 & 1 & 0 & 6.1 & 1.2 & 87.5 \\
-0.6 & 1.4 & 0 & 1.7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -0.6 & 0 & 0.1 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 2.3 \\
0.3 & 0.8 & 0 & 1.6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.1 & 0.1 & 0.1 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0.2 & 0 & 2.1 \\
-0.8 & 0.5 & 0 & -0.9 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1.4 & 0 & 0 & 0.1 & 0 & 0 & -0.1 & -0.1 & 0 & 0 & 0 & 0.1 & 0 & 0.1 \\
\hline
8.7 & 59 & 0 & 59.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.2 & 2.2 & 6.2 & 4.3 & 1.7 & 0 & 2.7 & 0.8 & 0 & 0 & 0 & 10.1 & 1.8 & 115.7
\end{array}\right]
$$

Løsningen er da $\overrightarrow{x} = [0, 0, 11.1, 0, 8.7, 7.5, 6.1, 0.1, 2.1, 10.8, 2.3, 0]^T$, og den optimale værdi af objektfunktionen er $z = 115.7$.

De indgange, der er blevet pivoteret igennem simplexalgoritmen i kronologisk rækkefølge er: $a_{1,10}$, $a_{2,7}$, $a_{12,8}$, $a_{4,5}$, $a_{3,3}$, $a_{10,11}$, $a_{11,2}$, $a_{6,1}$, $a_{11,9}$, $a_{7,12}$, $a_{6,6}$ og til sidst $a_{7,22}$.

7 Diskussion

7.1 Resultater

Den maksimale værdi af objektfunktionen fra det linære optimeringsproblem 6.1 er bestemt til 115.7 ved brug af simplexalgoritmen. Vi er meget tilfredse med resultatet, da det gav den optimale løsning, og koden bestemte den meget hurtigt. Det er svært at sammenligne de to algoritmer, da der kun er et resultat fra den ene, dog vil vi alligevel prøve.

7.2 Metode

Der er blevet beskrevet to algoritmer og anvendt én i projektet, derfor vil der i dette afsnit blive diskuteret forskellene på de to, og hvad fordelen er ved den ene frem for den anden.

7.2.1 Anvendelighed

Når man snakker anvendelse, er det lidt vanskeligt, da man aldrig rigtig ville bruge ellipsoidemetoden i virkeligheden, da den er upraktisk og brydsom. Fra et teoretisk synspunkt er ellipsoidemetoden dog yderst interessant, da den tager udgangspunkt et helt andet sted end simplexalgoritmen, nemlig mere geometrisk. Desuden har ellipsoidemetoden en polynomiel kompleksitet, hvilket er langt bedre end simplexalgoritmens kompleksitet. Dog er simplexalgoritmen altid at foretrække, når vi har med reelle problemer at gøre, da den løser de fleste problemer yderst hurtigt og problemfrit. Det er kun sjældent, at der opstår komplikationer, og når de gør, findes der metoder til at løse komplikationen, hvor blandt andet Blands algoritme kan anvendes.

7.2.2 Generalitet

De algoritmer der er blevet opsat igennem projektet, kan benyttes til at løse ethvert lineært programmeringsproblem. Dette er blevet demonstreret i de forskellige eksempler, hvor de forskellige konflikter, der kan opstå i simplex er blevet afklaret. Det kræver dog, at problemet er på standardform.

7.3 Validitet og reliabilitet

Validiteten af de to algoritmer er, at simplexalgoritmen bestemmer den optimale værdi til et lineært optimeringsproblem, mens ellipsoidealgoritmen kun tilnærmer sig. Dog er tilnærmelsen meget tæt på den optimale værdi til det lineære programmeringsproblem, og kan komme arbitrært nær den optimale værdi. Hvis algoritmerne gennemløber et lineært optimeringsproblem flere gange, vil den hver gang nå frem til det samme resultat.

8 | Konklusion

Lineære programmeringsproblemer er en gruppe af problemer, hvor det ønskes at optimere en lineær funktion, givet forskellige begrænsende ligninger og uligheder. En ligning og en funktion kaldes lineær, hvis alle led kun har én variabel, som er ganget på en koefficient. Yderligere skal begrænsningerne til et sådan problem også være på lineær form. For at bestemme en løsningsmetode til problemer på denne form, har det været fordelagtigt at benytte lineær algebra, og derfor er de relevante dele af lineær algebra blevet beskrevet. Dermed kan ligningerne og ulighederne i problemet opstilles i en matrix, hvorefter det bliver let at addere eller subtrahere uligheder fra andre uligheder, samt at multiplicere en dem. Til dette er der i projektet defineret en såkaldt pivotregel, som foretager rækkeoperationer således, at den udvalgte variabel kun indgår i én af ulighederne.

Selve løsningsmetoden, som beskrives i projektet, er simplexalgoritmen, der anvender pivoteringer i bestemte indgange. Disse udvælges af algoritmen, og sikrer at en konstant i den funktion, som det ønskes at maksimere, enten øges eller forbliver samme værdi for hver pivotering. Metoden beskrevet i dette projekt tager som input et problem på hvad der kaldes standardform. Hvis det aktuelle problem ikke er på standardform, kan det omdannes, ved at dele en lighedbegrænsning op i to uligheder, en $\leq$-ulighed og en $\geq$-ulighed. Derudover kan alle $\geq$-uligheder vendes ved at gange de ulighederne igennem med -1. Hertil skal det nævnes at ethvert minimeringsproblem kan løses ved at have problemet som dualen, og derefter løse problemets primal.

Til at kunne sammenligne simplexalgoritmen, er der i projektet også blevet undersøgt ellipsoidealgoritmen, der dog i praksis ikke benyttes. Denne algoritme laver to gæt på en optimal løsning til problemet, en som er mulig, men ikke nødvendigvis optimal, og en som er ikkemulig. Herefter indkredser algoritmen sig på et gæt på den optimale løsning. Algoritmen stopper efter et antal iterationer basereret på antallet af variable, samt størrelsen af den afvigelse som er acceptabel. Kompleksiteten af ellipsoidealgoritmen er polynomiel, hvilket er bedre end simplexalgoritmen, der har en eksponentiel kompleksitet. I praksis benyttes simplexalgoritmen som regel, da ellipsoidealgoritmen altid itererer som dens worstcasekompleksitet, mens simplex i mange tilfælde itererer færre gange end dens eksponentielle worstcase-kompleksitet.

Som en del af projektet er simplexalgoritmen blevet kodet i Python3, og et problem med 12 variable er blevet løst ved brug af denne. For dette er løsningen blevet bestemt til $\overrightarrow{x} = [0, 0, 11.1, 0, 8.7, 7.5, 6.1, 0.1, 2.1, 10.8, 2.3, 0]^T$, og den optimale værdi af objektfunktionen er $z = 115.7$, og den fandt dette resultat på 0.004 sekunder.

9 Perspektivering

I dette projekt har der været fokus på simplex- og ellipsoidealgoritmen, men det er kun simplexalgoritmen, der er blevet programmeret. En oplagt tilføjelse til projektet ville have været en programmering af ellipsoidealgoritmen, så en bedre sammenligning ville kunne laves. Dette var dog ikke muligt, grundet sværheden af ellipsoidealgoritmen, samt den relativt korte tidsramme for projektet. Udover det store arbejde dette ville være, så er der nogle teoretiske problemer, såsom bestemmelsen af R og r.

Desuden fokuserede projektet eksklusivt på lineær programmering, men der er et stort område af ikke-lineær programmering, som ikke blev undersøgt, da vi valgte at fokusere på lineær programmering. Dette resulterede også i, at mange algoritmer til løsningen af ikke-lineære optimeringsproblemer ikke kunne undersøges. De mest brugte algoritmer fungerer ved at omdanne lineære programmeringsproblemer til ikke-lineære programmeringsproblemer. Disse er hurtigere end simplexalgoritmen, og har en lavere kompleksitet, men er uden for projektets rammer.[Robere, 2012]

Under beskrivelsen af simplexalgoritmen kunne vi have været mere dybdegående, specielt når det kommer til den geometriske fortolkning af simplexalgoritmen, samt algoritmens kompleksitet. Dette er dog ekskluderet, da vi valgte at gå i en mere algebraisk retning. Ved ellipsoidealgoritmens afsnit blev en del mere undladt, da algoritme er mere vanskelig end simplexalgoritmen, så meget af teorien og implementationen bag den blev udeladt.

10 | Bibliografi

Bertsimas, D. and Tsitsiklis, J. [1997], *Introduction to linear optimization*, Athena Scientific, Belmont, Massachusetts. **ISBN**: 1-886529-19-1. 12

Bubeck, S. [2013], 'Orf523: the ellipsoid method'.
URL: *https://blogs.princeton.edu/imabandit/2013/02/07/orf523-the-ellipsoid-method/* 44

Camarena, O. A. [2016], 'Duality theorems matrix formulas for dictionaries'.
URL: *https://www.matem.unam.mx/ omar/math340/index.html* 28

CS-149 Staff [2007], 'Notes on simplex algorithm'.
URL: *http://cs.brown.edu/courses/csci1490/notes/day9.pdf* 30

Ekeocha, R. J. O., Uzor, C. and Anetor, C. [2018], 'Paper- the use of the duality principle to solve optimization problems'.
URL: *https://online-journals.org/index.php/i-jes/article/view/8224* 15

Geil, O. [2015], *Elementary linear algebra 2015, Second edition*, Pearson. **ISBN**: 978-1-78448-2. 3

Hurlbert, G. [2010], *Linear Optimization. The Simplex Workbook*, Springer. **ISBN**: 978-0-387-79147-0. 2, 12, 17

Israel, R. [2006], 'Notes on the simplex method from sept. 18 class, phase i: Artificial variable method', University of British Columbia. 22

Kalai, G. and Kleitman, D. [1992], 'Quasi-polynomial bounds for the diameter of graphs of polyhedra', Bull. Amer Math. Soc. 1, 33

Math. Dept. [2016], 'Linear optimization, lecture 12'.
URL: *https://sites.math.washington.edu/ burke/crs/407/lectures/L12-w10.pdf* 30

Matoušek, J. and Gärtner, B. [2007], *Understanding and Using Linear Programming*, Springer. **ISBN**: 3-540-30697-8. 33

MIT [18/03/2016], 'Mit, 6.854 spring 2016 lecture 12: From separation to optimization and back; ellipsoid method'.
URL: *https://www.youtube.com/watch?v=qWpYtfXeEn8* 40

Newman, A. [2016], 'Optimization and approximation. lecture 4', ENS Lyon. 30, 33

O'Donnell, R. [2011], 'Lecture 8 the ellipsoid algorithm'.
URL: *https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture08.pdf* 42

Robere, R. [2012], 'Interior point methods and linear programming'.
URL: *https://www.cs.toronto.edu/ robere/paper/interiorpoint.pdf* 50

The Royal Swedish Academy of Sciences [2019], 'Press release on the 1975 nobel prize in economics'.
URL: *https://www.nobelprize.org/prizes/economic-sciences/1975/press-release* 1

UIO [20/01/2014], 'A mini-introduction to convexity'.
URL: *https://www.uio.no/studier/emner/matnat/math/MAT-INF3100/v14/convmat-inf3100.pdf* 34

Walker, M. [2017], 'Convex analysis'.
URL: *http://www.u.arizona.edu/ mwalker/econ519/Econ519LectureNotes/ConvexAnalysis.pdf* 36

A Appendix

Her er den kode, som er anvendt i løbet af rapporten.

A.1 Matrix koden

Framework kode.

```

class Matrix:                         #Definer for python hvad en matrix er, lister af lister
    """docstring for Matrix."""
    def __init__(self, matrix):
        self.matrix = [Vector(row) if type(row) != Vector else row for row in matrix]
        self.m = len(matrix)          #m til at vaere antallet af raekker

        self.n = len(matrix[0])       #Antallet af indgange i den foorste raekke
        for row in self.matrix:             #kigger igennem alle raekkerne
            if len(row) != self.n:

                raise "Invalid matrix input!"

        self.rows = self.matrix             #raekkerne er defineret godt ved brug af listerne
        self.columns = []             #soejler skal defineres

        # Inefficiently map out the columns of the matrix
        for i in range(self.n):       #for alle r kker l g den i te indgang til kolonnen
            column = []
            for row in self.rows:     #en ny matrix der best r af dens transponerede
                column.append(row[i])
            self.columns.append(Vector(column))

    def __iter__(self):               #itererer over
        """ Iterate over the rows by default """
        yield self.matrix             #giv resultat og vente, spytter listerne ud, foorst foorste raekke osv.

    def __getitem__(self, key):
```

```
        return self.rows[key]

    def __len__(self):
        return len(self.rows)

    def __repr__(self):              #naar der skrives print M kan man printe en class, s  det er laeseligt
        out = "<Matrix with %g rows and %g columns: \n" % (self.n, self.m)
        for row in self.rows:        #hvor mange raekker og soojler er der og se den.
            out += str(row)
            out += '\n' if row != self.rows[-1] else ''  #lave det til en string
        out += '>'
        return out

    def __mul__(self, number):       #definere hvordan det kan ganges sammen
        if type(number) == type(self): #sammenlign de to, hvis begge er matrixer gaaes til matrixprodukt
            return self.matrixProduct(number)

        out = []
        for n in range(self.n):      #definere hvordan matricer og numre ganges sammen
            row = []
            for m in range(self.m):
                row.append(self.matrix[n][m] * number)
            out.append(row)
        return Matrix(out)

    def __rmul__(self, number):      #definer at 3*M er det samme som M*3
        return self.__mul__(number)

    def __add__(self, other):        #laegger til matrixen
        if type(other)!= Matrix:     #begge skal vaere matricer for det duer
            print("Cannot implicity add/subtract matrix and non-matrix!")
            return False

        if (self.m, self.n) != (other.m, other.n):  #de skal have samme dimention
            print("Cannot add matrices of different sizes!")
            return False

        out = []
        for n in range(self.n):
            row = []
            for m in range(self.m):
                row.append(self.matrix[n][m] + other.matrix[n][m])  # TODO: round andet sted
            out.append(row)
        return Matrix(out)
```

```
    def __sub__(self,other):          # shortcut: ganger med -1
        return self.__add__(-1*other)

    def stitch_matrix_right(self, other_matrix): #saette to matricer sammen
   til hoojre for f eks naar A og I_s skal saettes sammen :)

        mat = []     # ny matrix hvor de er sat sammen

        for own_row, other_row in zip(self.rows, other_matrix.rows):

            mat.append( own_row % other_row )

        return Matrix(mat)

    def stitch_vector_right(self, vector):
        """ add a vector to the right side of self
        and return a new matrix object."""

        mat = []     # ny matrix hvor de er sat sammen

        for own_row, value in zip(self.rows, vector):

            own_row.append(value)
            mat.append( own_row)

        return Matrix(mat)

    def add_row(self, vector):
        """takes a vector and returns a new Matrix object
        of the self with added vector. Raises ValueError
        if the length of the vector is different from n"""

        while len(vector) < self.n:
            vector.append(0)


        if len(vector) != self.n:
            raise ValueError("Can't add vector, dimension are off.")

        rows = self.rows
        rows.append(vector)

        return Matrix(rows)

    def transpose(self):
        """returns a Matrix of the transposed"""

        return Matrix(self.columns)

    def set_row(self, row_number, incoming_vector):
```

```
        """
        row_number:       index 0 <= row < n
        incoming_vector: vector to change to

        Set the row at row number to incoming vector
        does not return a new Matrix object.

        Raises TypeError if row number isnt an int
        and if incoming vector isn't a vector
        """

        if type(row_number) != int or type(incoming_vector) != Vector:
            raise TypeError("Input %s, %s can't be row-set" % (row_number,
    incoming_vector))

        row0 = self.rows[row_number]
        if len(row0) != len(incoming_vector):
            print("Can't change dimensions of row")
            return False
        # ------------------------ #

        self.rows[row_number] = incoming_vector

    def elementary_operation(self, emo):
        """ perform the elementary operation
        by row0 = row0 + row1 * mult
        EMO = (index row0, index row1, mult)

        raises TypeError if emo isnt class EMO
        """
        if type(emo) != EMO:
            raise TypeError("Can't perform elementary row operation with non
    emo")
        row0 = self.rows[emo.row0]
        row1 = self.rows[emo.row1]


        self.set_row(emo.row0, row0 + row1 * emo.mult)
        return True

    def column_pivot(self, row, column):
        """performs a column pivot on the matrix

        takes a (row, column), reduces the value to 1
        and reduces all other entrances in the row
        to 0 and returns self.

        raises TypeError if row or column isnt int.
        raises KeyError if row > n or column > m.
        """

```

```
        if type(row) != int or type(column) != int:
            raise TypeError("Can't find index (%s,%s)" % (row, column))

        if row > self.n or column > self.m:
            raise KeyError("Index out of range")

        # Make sure, the pivot row has leading digit 1
        pivot_row = self.rows[row]
        pivot_entrance = self.rows[row][column]

        emo = EMO(row, row, mult = 1 / pivot_entrance - 1)

        self.elementary_operation(emo)

        pivot_row = self.rows[row]
        # Reduce all other entrances in column to 0
        for c, row in enumerate(self.rows):
            if row == pivot_row: continue;
            self.rows[c] = self.rows[c] - pivot_row * row[column]

        return self

    def VectorProduct(self, Vector):
        """compute the vector product of self * vector"""

        if len(Vector) != self.n:
            return False

        out = []
        for row in self.rows:
            out.append(round(row * Vector,1))
        return out

    def matrixProduct(self, other):
        if type(other) != Matrix:
            print("use <A * B> to multiply a matrix by a number")
            return False

        if self.n != other.m:
            print("To multiply matrices, they need form m x n and n x p")
            return False

        out = []

        for column in other.columns:
            out.append(self.VectorProduct(column))

        return Matrix(out)

    def mvp(self, Vector):
        return self.VectorProduct(Vector)
```

```

    def mmp(self, other):
        return self.matrixProduct(other)

    def smr(self, other):
        return self.stitch_matrix_right(other)

    def svr(self, other):
        return self.stitch_vector_right(other)

class Vector:
    """docstring for Vector."""
    def __init__(self, vector):
        if type(vector) != tuple and type(vector) != list:

            raise "Can not interpret this as a vector!"
        self.values = vector

    def __iter__(self):
        for value in self.values:
            yield value

    def __getitem__(self, key):
        return self.values[key]

    def __repr__(self):

        return str([round(elem, 2) for elem in self])

    def __len__(self):
        return len(self.values)

    def __getitem__(self, ind):
        return self.values[ind]

    def __add__(self, other):

        if type(other) != type(self): return False;
        if len(other) != len(self): return False;

        values = [v1 + v2 for v1, v2 in zip(self, other)]
        return Vector(values)

    def __mod__(self, other):
        return Vector(self.values + other.values)

    def __eq__(self, other):
        if type(other) != type (self): return False;
        return self.values == other.values

    def __mul__(self, number):
```

```
        if type(number) == type(self):
            if not len(number) == len(self):
                print("Can't multiply Vectors of differnet length")
                return False

            return sum([v1 * v2 for v1, v2 in zip(self, number)])

        return Vector([ v * number for v in self ])

    def __rmul__(self, number):
        return self.__mul__(number)

    def __sub__(self, other):
        return self.__add__(-1*other)

    def __rsub__(self,other):
        return self.__sub__(other)

    def append(self, val):
        self.values.append(val)
        return Vector(self.values)


class EMO:
    """ docstring for EMO """
    def __init__(self, row0, row1, mult=1):
        self.row0, self.row1, self.mult = row0, row1, mult
        self.emo = (row0, row1, mult)

    def __repr__(self):
        return "<Elementary operation: row0:{}, row1:{}, mult:{}>".format(self.row0, self.row1, self.mult)

    def __getitem__(self, key):
        return self.emo[key]






# Find lowest argument in list, and return the index of said argument
def argmin(v, greaterthan = False):
    #index = False
    if greaterthan is False:
        hold = max(v)
        for c, val in enumerate(v):
            if val < hold:
                hold = val
                index = c
    else:
```

```python
            hold = max(v)
            for c, val in enumerate(v):
                if val <= hold and val > greaterthan:
                    hold = val
                    index = c
    return index

def main():
    pass

if __name__ == '__main__':
    main()
else:
    eps = 1E-14
```

A.2 Simplexkoden

Simplexalgoritmen skrevet ind i Python. Den benytter sig af frameworket i A.1.

```python

""" import - notation and classes from matrix.py """
from matrix import *

def identity_matrix(n):
    """function to make an n-degree identity mat-
    rix, and return the matrix object """

    mat = [[1 if j == i else 0 for j in range(n)] for i in range(n)]

    return Matrix(mat)



# Returns tuple of
#                [0]: boolean for succes
#                [1]: error message
def curate(object_vector, constraints, weight_vector):
    """takes the input given to the simplex and
    curates it, raises appropriate errors
    """

    # Test if all input types are lists
    types = [True if type(obj) == list else False for obj in [object_vector, constraints, constraints[0], weight_vector]]
    if not all(types):
        raise TypeError("All input types must be lists, and constraints must be a nested list.")


```

```
    constraints_n = len(constraints[0])

    # Tests if the dimensions are
    if len(object_vector) != constraints_n:
        raise ValueError("Object vector must have same amount of variables as constraints. If a variable doesn't exist in a constraint, add a 0.")

    if len(constraints) != len(weight_vector):
        raise ValueError("Each constraint must have a weight.")

    return True


def simplex(object_vector, constraints, weight_vector):
    """
    object_vector is an list of coefficients
    constraints is a list of list of coefficients
    weight_vector is a list of coefficients

    takes a maximization problem on tableux form
    and uses the simplex method, combined with b
    lands pivot rule, to optimize the set of li-
    near equations. """


    # Clean the input
    curate(object_vector, constraints, weight_vector)


    print(
    """Starting simplex on
    Object vector: {}
    Constraints: {}
    Weight vector: {}
    """.format( object_vector, constraints, weight_vector ))

    object_vector = Vector(object_vector)
    constraints = Matrix(constraints)
    weight_vector = Vector(weight_vector)

    object_value = 0

    #object_value.append()

    # Make sure the amount of constraints == amount of weights
    if len(constraints) != len(weight_vector):
        raise 'amount of restraints does not match amount of contraints'

    # Make an identity matrix, one slackvariable per constraint
    slack_variables = identity_matrix(len(constraints))

```

```
    print("Setting up tableux...\n")
    # Set up tableux
    tableux = constraints.smr(slack_variables)
    tableux = tableux.svr(weight_vector)
    tableux = tableux.add_row(-1*object_vector)

    while True:
        # Rule 1: If all values in the object vector is positive, the solution is found.
        positive_object = [v >= -eps for v in tableux[-1]] # epsilon instead of 0 because rounding error

        if all(positive_object):
            return (tableux[-1][-1], tableux, "ERR100: Problem is optimal")

        # Find lowest coefficient in object vector
        relevant_column = argmin(tableux[-1][:-1])

        # Compute ratios between contraints in relevant_column and weights
        ratios = [row[-1] / row[relevant_column] for row in tableux[:-1]]

        # Rule 2: If all ratios are negative, problem is unbounded
        positive_ratios = [False if v < 0 else True for v in ratios]
        if not any(positive_ratios):
            return (False, None,"ERR200: Problem is unbounded")

        # Rule 3: Pivot in the relevant cell
        # Find the lowest ratio greater than 0
        relevant_row = argmin( ratios, 0)

        # Pivot in this variable
        print("Pivoting in ({}, {})..." .format(relevant_row+1, relevant_column+1))
        tableux = tableux.column_pivot(relevant_row, relevant_column)
        #print(tableux)



def main():
    pass

if __name__ == '__main__':
    main()
```

A.3 Eksempel på koden

Et eksempel på, hvordan simplexkoden kan benyttes til at løse et problem. De valgte problemer er (4.3) og (4.19).

```
""" docstring for example simplex problem
    has 2 different sample problems

    - working_example
    - unbounded_example
"""

from simplex_algorithm import *

def working_example():
    """
    a working example of the simplex algorithm
    applied in python

    z = 17x_1 + 70x_2 + 13x_3
        2x_1  -  6x_2 -  4x_3   <= -4
        -x_1  -  2x_2 -  3x_3   <= -2
        4x_1  +  2x_2 -  1x_3   <=  3
        3x_1  -  3x_2 +  3x_3   <= -1
        x_1, x_2, x_3          >=  0
    """

    # Insert an object vector on list form
    object_vector = [
    17, 70, 13
    ]

    # Insert the constraints on nested list form
    constraints = [
    [ 2, -6, -4],
    [-1, -2, -3],
    [ 4,  2, -1],
    [ 3, -3,  3]
    ]

    # Insert the weight vector on list form
    weight_vector = [
    -4,
    -2,
    3,
    -1
    ]

    return object_vector, constraints, weight_vector

def unbounded_example():
    """
    an example of of the simplex algorithm
    when the basic solution is unbounded

    z =  2x_1 - 3x_2 - 3x_3
```

```
        -3x_1 + 2x_2          <= 80
        - x_1 + 1x_2 + 4x_3   <= 20
        -2x_1 - 2x_2 + 5x_3   <= 30
        x_1, x_2, x_3         >=  0
    """

    # Insert an object vector on list form
    object_vector = [2, -3, -3]

    # Insert the constraints on nested list form
    constraints = [[-3, 2, 0], [-1, 1, 4], [-2, -2, 5]]

    # Insert the weight vector on list form
    weight_vector = [80, 20, 30]

    return object_vector, constraints, weight_vector

# When file is run, execute the example
def main(problem):
    #print(problem)
    object_vector, constraints, weight_vector = problem()

    value, tableux, statement, pivots = simplex(object_vector, constraints,
    weight_vector, silent=False, step = True)

    if not statement.startswith('ERR100'):
        print(statement)
        return

    print("\nReturn statement: %s \n" % statement)
    print("Solved tableux for the problem: \n %s \n" % tableux)
    print("Took %s pivots to reach optimal point" % pivots)
    print('The maximum value of the problem: %s \n' % round(value,2))

if __name__ == '__main__':
    import sys

    if len(sys.argv) > 1:
        main(locals()[sys.argv[1]])
    else:
        print("""Please specify which example to run;
        - working_example
        - unbounded_example""")
```