Umeå Universitet
Institutionen för matematik och matematisk statistik February 11, 2020

Linjär Programmering - Passé simplex?

Alexander Gunillasson (alexander.gunillasson@student.umu.se)
Aladdin Persson (aladdin.persson@student.umu.se)
Simon Edman (sied0005@student.umu.se)
Joel Eliasson (joel0089@student.umu.se)

Laboration 2
Handledare: Lars-Daniel Öhman

Contents

1 Opponering

Vår granskningsgrupp tyckte överlag att vår rapport var väldigt bra men vi har valt att göra om och utvecklat olika delar av vår rapport efter att ha tagit åt oss av deras kommentarer (se Appendix, 8.7). Vi valde bland annat att tydliggöra vår definition av målfunktionen, z i *Analytisk lösning för n=2.* Vidare så önskade opponenterna att vi förtydligade radoperationerna vilket vi har åtgärdat. Vi hade även en del rubriknamn som de önskade att vi omformulerade. Efter att vi hade gjort det var det lättare att förstå vad tabellen/figuren ville förmedla. De ville även se en tydligare koppling till vår teoridel vilket vi har försökt göra klargöra. Vidare har vi även försökt förtydliga definitionerna av olika variabler för att förebygga missuppfattningar. Till exempel har vi i *Utförande* försökt klargöra att j utför en rad. I *Resultat* fanns det en del oklarheter gällande hur vi kunde erhålla decimaltal, men det har vi förtydligat. Avslutningsvis har vi även justerat så att ett stycke som vi från början hade i *Analytisk lösning för n=2* istället återfinns i *Diskussion.*

2 Introduktion

Simplexmetoden är idag den mest använda algoritmen för att lösa LP-problem och uppfanns av George Dantzig 1946. Metoden är i praktiken en mycket effektiv algoritm med polynomisk tid, däremot finns det fall där simplexmetoden kräver exponentiell tid. Ett sådant exempel introducerades 1972 av Klee och Minty och visade därmed att sämsta fallet för simplexmetoden kräver exponentiell tid. I denna rapport vill vi undersöka empiriskt hur effektiv simplexmetoden är och undersöka effektivitet när den körs med den besvärliga familjen introducerat av Klee och Minty.

Simplexmetoden finns även i olika implementationer i hur man väljer stegriktning. I denna rapport undersöker vi tre olika implementationer som vardera använder olika regler för val av stegriktning. Vi undersöker dels den vanliga simplexmetoden, Blands regel och en som kollar på förbättring av målfunktionsvärdet. Detta beskrivs i mer detalj under teoridelen. I rapporten kollar vi även på en analytisk lösning för mindre fall av Klee-Mintys problem med den vanliga simplexmetoden och undersöker geometrisk varför detta kräver lång tid. Slutligen visar vi i tabellform och i grafer hur de olika implementationerna av simplex resulterar i olika antal iterationer och tid som krävs för att få fram en lösning.

3 Teori

Simplexmetoden existerar i flera olika versioner och vi vill undersöka i mer detalj hur dessa fungerar. Samtliga utgår från att LP-problemet är skrivet i standardform och att vi har en baslösning. Följaktligen väljer vi att definiera dessa begrepp nedan.

Definition 1 *Ett allmänt LP-problem skrivet på standardform kan formuleras enligt*

$$\begin{aligned} &max\ z = \sum_{j=1}^{n} c_j x_j \\ &då\ \sum_{j=1}^{n} a_{ij} x_j = b_i,\ \ i = 1, ..., m \\ &x_j \geq 0,\ \ j = 1, ..., n \end{aligned}$$

Detta kan skrivas på kompakt form som

$$\max \left\{ c^T x \mid Ax = b,\ x \geq 0_n,\ x \in R^n \right\}. \tag{1}$$

Definition 2 *En baslösning till ekvationssystemet* $\boldsymbol{Ax = b}$ *erhålls om n - m (n= variabler, m= bivillkor) variabler sätts till 0 och resterande variabler får de unika värden som erhålls då det kvarvarande ekvationssystemet löses. De variabler som sätts till 0 kallas för icke-basvariabler och de resterande m variablerna kallas basvariabler.*

Eftersom vad som definieras som reducerande kostnad kan variera väljer vi att definiera för att förtydliga.

Definition 3 *Då vi vill maximiera målfunktionsvärde z*

$$z = c_1 x_1 + c_2 x_2 + ... + c_n x_n.$$

Väljer vi att kalla $c_1, ..., c_n$ *som positiva reducerande kostnader. Och motsvarande*

$$z - c_1 x_1 - c_2 x_2 - ... - c_n x_n = 0$$

som är formen skriven i simplextablån med negativa reducerande kostnader.

3.1 Varianter av simplexmetoden

I många fall hänvisas simplexmetoden i bestämd form som i fallet att det endast finns ett entydigt sätt att implementera den på. Detta är förståeligt om man antar att man hänvisar till den vanliga simplexvarianten. Därefter har det uppkommit flera varianter som fungerar bättre på vissa exempel och därmed finns det inte ett enda sätt att implementera simplex, utan det finns flera olika varianter. Vi har valt ut tre vanliga, bland dessa den vanliga simplexmetoden och två andra som fungerar bättre på vissa exempel.

Algoritmen för samtliga metoder för simplex utgår från att vi har ett linjärt program skrivet i en simplextablå som är på standardform. Algoritmerna startar med hörnpunkt som är inom det tillåtna området och som är en baslösning. Idén är senare till följd av att vi vet att varje optimal lösning för ett LP-problem återfinns i en hörnpunkt, genomföra basbyten som gör att vi hoppar mellan hörnpunkterna. Det är dessa basbyten som kan göras på flera olika sätt, d.v.s det finns flera sökriktningar för att förbättra sin nuvarande målfunktion. Dessa bygger på olika heuristik och är det som gör att simplexmetoden kan skilja sig beroende på implementation. För samtliga gäller det att om inga reducerande kostnader är negativa är lösningen optimal och detta är därmed ett avbrottskriterium. Samt gäller det att om alla koefficienter för respektive bivillkor, framför den inkommande variabeln x_k är icke-positiva så har problemet en obegränsad lösning[1].

3.1.1 Vanlig simplex

Den vanliga simplexmetoden brukar kallas att använda sig av steepest ascent eftersom den kollar på vilken inkommande variabel som gör störst påverkan på målfunktionsvärdet. Sökriktningen eller basbytet för vanliga simplex utgår från simplextablån där man väljer den inkommande variabeln x_k som den mest mest positiva reducerande kostnaden och därefter väljs den utgående variabel genom att undersöka ett kvottest, där den lägsta kvoten finns av intresse. Detta utförs genom att kolla på bivillkor i, och delar med koefficienten framför x_k och gör detta för samtliga bivillkor. Sedan tar man det minsta av dessa och om flera skulle ha samma kvot tar man den med minst index i. Detta kan skrivas

ekvivalent som

$$\min_i \frac{\widetilde{b_i}}{\widetilde{a_{ik}}}. \tag{2}$$

Då blir inkommande variabeln x_k och utgående variabel x_i.

3.1.2 Blands regel

För Blands regel väljs den inkommande variabeln x_k som den som har lägst index bland icke-basvariablerna med positiv reducerande kostnad. Detta är i kontrast till vanliga simplex som väljer den som har mest positiva reducerande kostnaden. Utgående variabeln x_i väljs på liknande sätt som i vanliga simplex med den som har lägst kvot och lägst index.

3.1.3 Sökriktning för bästa målfunktionsvärde

Den som ger störst förbättring av målfunktionens värde. Detta skiljer sig från den "vanliga" regeln, genom att den vanliga regeln bara tittar på lutningen på förbättringen, inte hur stor förbättringen blir totalt. Det kan ju vara stor lutning, men kort väg.

3.2 Klee och Minty problem

Den besvärliga familjen av instanser som introducerades av Klee och Minty 1972 definieras som följande

$$\begin{aligned} &\text{maximiera } z = \sum_{i=1}^{n} 2^{n-i} x_i \\ &\text{då } \sum_{i=1}^{j-1} 2^{j-i} x_i + x_j \leq 5^j, \ j = 1, ..., n \\ &\qquad x_j \geq 0, \ j = 1, ..., n. \end{aligned} \tag{3}$$

4 Analytisk lösning för n = 2

Baserat på Klee-Mintys exempel som vi tagit upp i Teoridelen undersöks nedan problemet för $n = 2$. Vi börjar med att illustrera problemet utifrån en skiss med tillhörande extrempunkter av det tillåtna området vilket *Figure* 1 illustrerar.

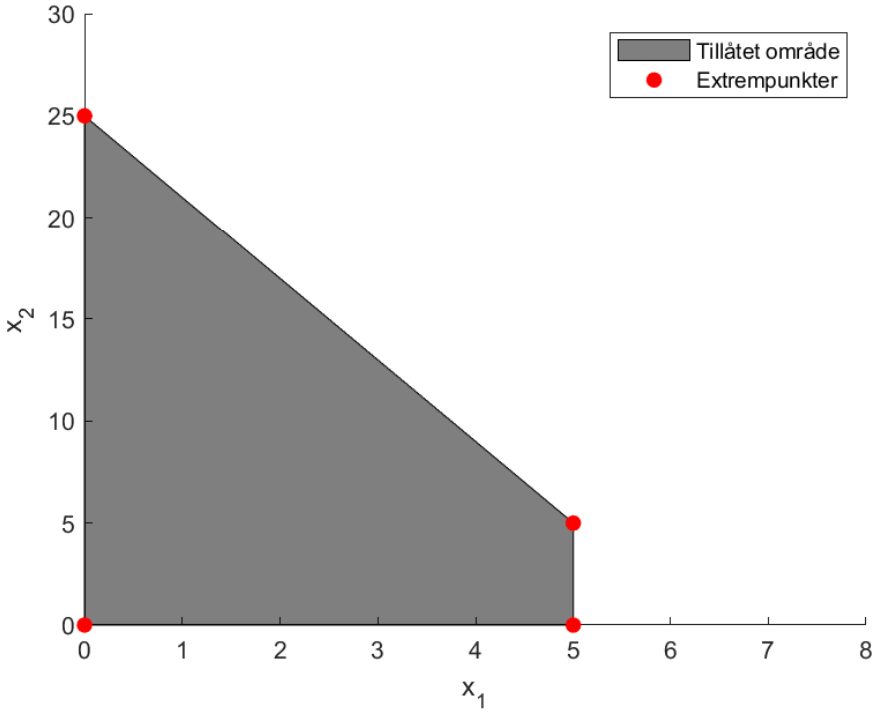

Figure 1: Det tillåtna området för problemet när $n = 2$.

Ursprungligen erhåller vi följande LP-problem som vi vill lösa då vi sätter $n = 2$.

$$\begin{array}{lrcl} \max & \mathrm{z} = & 2x_1 + x_2 & \\ \text{då} & x_1 & \leq & 5 \\ & 4x_1 + x_2 & \leq & 25 \\ & x_1, x_2 \geq & 0 & \end{array}$$

Vidare väljer vi att skriva om uttrycket på standardform som hittas i teori, *Definition* 1. Då erhålls följande:

$$
\begin{array}{lrl}
\max & \mathrm{z} = & 2x_1 + x_2 \\
\text{då} & x_1 + s_1 & = 5 \\
& 4x_1 + x_2 + s_2 & = 25 \\
& x_1, x_2, s_1, s_2 \geq & 0
\end{array}
$$

Följaktligen behöver vi välja basvaribler respektive icke-basvariabler för att kunna skapa en baslösning. Vi använder vi *Definition* 2 från teoridelen, det vill säga n-m, vilket ger oss att $x_1 = 0$ och $x_2 = 0$ vilket innebär att dem blir icke-basvariabler. Således blir $s_1 = 5$ och $s_2 = 25$ våra basvariabler till baslösningen. Nu kan vi skapa en simplextablå, *Table* 1 för att lösa problemet samt finna vilka hörnpunkter som stegas igenom fram till lösningen. Innan vi ställer upp vår simplextablå så transformerar vi målfunktionen till ett målvillkor.

	z	x_1	x_2	s_1	s_2	Värden
z	1	-2	-1	0	0	0
s_1	0	1	0	1	0	5
s_2	0	4	1	0	1	25

Table 1: Ursprungliga simplextablån innan iteration 1, där den röda markerade rutan står för pivoteringsvärdet.

Först väljer vi pivoteringskolumnen som kolumnen med den största reducerade kostnaden. Vidare dividerar vi varje element i den sista kolumnen, *Värden* med motsvarande värde i pivoteringskolumnen för att avgöra det lägsta pivoteringsvärdet. Vi får att $5/1 < 25/4$, alltså väljer vi x_1 som inkommande variabler och s_1 som utgående. Avslutningsvis vill vi i första iterationen få nollor ovan och under pivoteringsvärdet. Det får vi genom att utföra radoperationer, alltså låter vi $(III) = (III) - 4(II)$ samt $(I) = (I) + 2(II)$. Nu har vi alltså rört oss från startpunkten, $(0, 0)$ till punkten $(5, 0)$ och erhåller en uppdaterad simplextablå, *Table* 2, vilken följer nedan.

	z	x_1	x_2	s_1	s_2	Värden
z	1	0	-1	2	0	10
x_1	0	1	0	1	0	5
s_2	0	0	1	-4	1	5

Table 2: Simplextablån innan iteration 2, där den röda markerade rutan står för pivoteringsvärdet.

Först väljer vi pivoteringskolumnen som kolumnen med den största reducer-

ade kostnaden. Vidare dividerar vi varje element i den sista kolumnen, *Värden* med motsvarande värde i pivoteringskolumnen för att avgöra det lägsta pivoteringsvärdet. Vi får att x_2 blir inkommande variabel medan s_2 blir utgående. Avslutningsvis vill vi i andra iterationen få nollor ovan och under pivoteringsvärdet. Det får vi genom att utföra radoperationer, vi låter $(I) = (I) + (III)$. Nu har vi alltså rört oss från punkten $(5, 0)$ till $(5, 5)$ och erhåller en uppdaterad simplextablå, *Table* 3, vilken följer nedan.

	z	x_1	x_2	s_1	s_2	Värden
z	1	0	0	-2	1	15
x_1	0	1	0	1	0	5
x_2	0	0	1	-4	1	5

Table 3: Simplextablån innan iteration 3, där den röda markerade rutan står för pivoteringsvärdet.

Först väljer vi pivoteringskolumnen som kolumnen med den största reducerade kostnaden. Vidare dividerar vi varje element i den sista kolumnen, *Värden* med motsvarande värde i pivoteringskolumnen för att avgöra det lägsta pivoteringsvärdet.Vi får att s_1 blir inkommande variabel medan x_1 blir utgående. Avslutningsvis vill vi i den tredje iterationen få nollor ovan och under pivoteringsvärdet. Det får vi genom att utföra radoperationer, alltså får vi $(I) = (I) + 2(II)$ samt $(III) = (III) + 4(II)$. Nu har vi alltså rört oss från punkten $(5, 5)$ till $(0, 25)$ och erhåller en uppdaterad, klar simplextablå, *Table* 4, vilken följer nedan.

	z	x_1	x_2	s_1	s_2	Värden
z	1	2	0	0	1	25
s_1	0	1	0	1	0	5
x_2	0	4	1	0	1	25

Table 4: Slutgiltiga simplextablån.

Alla reducerade kostnader är nu icke-positiva, vilket innebär att vi funnit en optimallösning där $x^* = (x_1, x_2) = (0, 25)$ samt att $z^* = 25$. För att finna den här lösningen behövde simplexmetoden som tidigare nämnt stegen igenom samtliga extrempunkter i följande ordning för att finna lösningen: $x = (0, 0), (5, 0), (5, 5), (0, 25)$.

5 Utförande

För att utföra denna laboration och testa olika implementationer av simplex metoden har vi använt oss av programvaran Matlab. Vi har valt att strukturera vårt program så att användaren kan välja vilken sökrikiktning simplex-metoden skall använda.

Den första implementationen vi gjorde var den besvärliga familjen av instanser. För att göra detta skapade vi en funktion som generar 3 stycken variabler som sedan kan tas in som parametrar till simplex metoden. Först tittar vi på målfunktionen från LP problemet definierat i Ekv.(3) som är en summa som går från i = 1 till n. Vi loopar igenom detta intervall och för varje iteration uppdateras en array "c" och det det nya värdet läggs in sist i arrayen. Därefter tittar vi på bivillkoren. Här har vi 2 stycken intervaller som vi måste ta hänsyn till, vi börjar med att skapa en matris och en vektor, matris "A" som är en n*n-matris kommer innehålla alla koefficienter hos bivillkoren medan vektor "b" kommer innehålla alla basvariabler. Vi loopar nu igenom intervallet j = 1 till n, vilket utgör en rad, och för varje rad fylls den kolumnvis på intervallet i = 1 till j - 1 där värdet på koefficienterna hos bivillkoren uppdateras och läggs in i matris A. Basvariablerna uppdateras för varje rad som gås igenom och läggs in i vektor b och diagonalen i matris A fylls med ettor. I slutändan har vi alltså en matris A vars rader består av koefficienter, en vektor b med basvariabler och en array c innehållande koefficienterna i målfunktionen. Detta resulterar i koden som finns i Appendix (8.2).

Nästa steg var att skapa en funktion som istället slumpar varje koefficient i standardformen utifrån en likformig diskret fördelning. Detta åstadkoms genom att en nedre och en övre gräns sätts. Vi använder i denna funktion exakt samma intervall som vi iterar över men skillnaden är att koefficienterna istället slumpas mellan de två gränserna. Exempelvis kan vi sätta en nedre gräns på 1 och en övre gräns på 500, därefter slumpas värden inom det intervallet.

Den ursprungliga simplexmetoden har vi fått given. Utifrån denna har vi sedan implementerat de två övriga sökmetoderna Blands regel och sökriktning för bäst målfunktionsvärde.

5.0.1 Blands regel

För att implementera Blands handlade det om att skriva om vanliga simplexmetoden så att inkommande variabeln blir den som har lägst index bland icke-basvariabler med negativt reducerad kostnad. Detta är egentligen enda skillnaden mot vanliga simplex, utgående variabeln väljs på samma sätt, det vill säga den som har lägst kvot och om flera har samma kvot väljs den med lägst index.

5.0.2 Sökriktning för bästa målfunktionsvärde

Man kollar på alla negativa reducerade kostnader och gör ett basbyte för utgående variabeln med den som har lägst kvot. För att genomföra det här undersökte vi hur mycket detta bytet skulle ha påverkat målfunktionen. Vidare utför vi det här för samtliga negativa reducerande kostnader. Det par av inkommande variabel och utgående som resulterar i bäst målfunktionsvärde blir således paret vi utför vår faktiska simplexiteration med och dessa sparar vi som "best_ind" och "best_pivot", där pivot är för raden och ind kolumnen. Kontrollen om den förbättrar målfunktionsvärdet görs genom att jämföra med "best_pivot". Mer specifikt för implementation, se i Appendix under kod för 'mysimplex_bestcost.m'.

6 Resultat

Alla de olika metoderna körs två gånger, en gång med Klee och Mintys besvärliga familj och en gång med slumpade tal från den diskreta likformiga fördelningen. Nedan tabeller redovisar resultatet i sekunder (s) och iterationer (i) och även tillhörande plot för att tydliggöra hur tiden, angivet i sekunder, förändras beroende av storlek på n. Iterationerna för respektive simplexmetod varierar vid körning med slumpade koefficienter och därför har vi valt att ta fram ett medelvärde för antalet iterationer baserat på 10 repetitioner.

n	Klee och Minty (s)	Random (s)	Klee och Minty (i)	Random (i)
2	0.00077721	0.00023766	3	2.400
4	0.00084204	0.00027606	15	3.400
6	0.0020825	0.00029979	63	3.400
8	0.0090332	0.00038262	255	4.400
10	0.03589	0.00044546	1023	4.600
12	0.13803	0.00057555	4095	5.200
14	0.54917	0.00067655	16383	6.600
16	2.3057	0.00084991	65535	7.400
18	9.3323	0.00094537	262143	5.200

Table 5: Vanlig simplex - steepest ascent

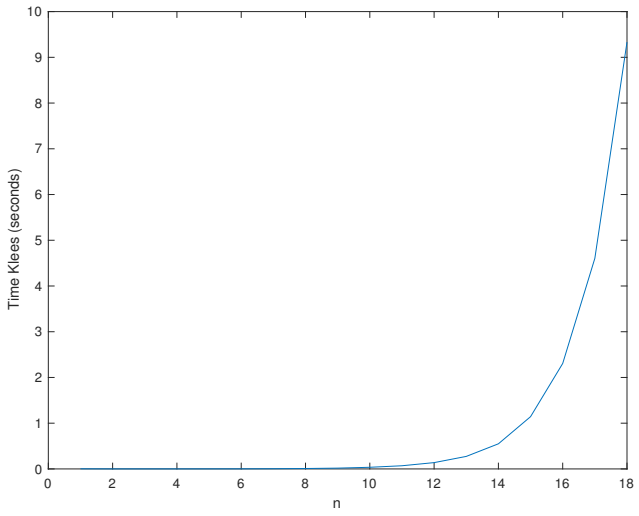

Figure 2: Vanlig simplex - Klee och Minty

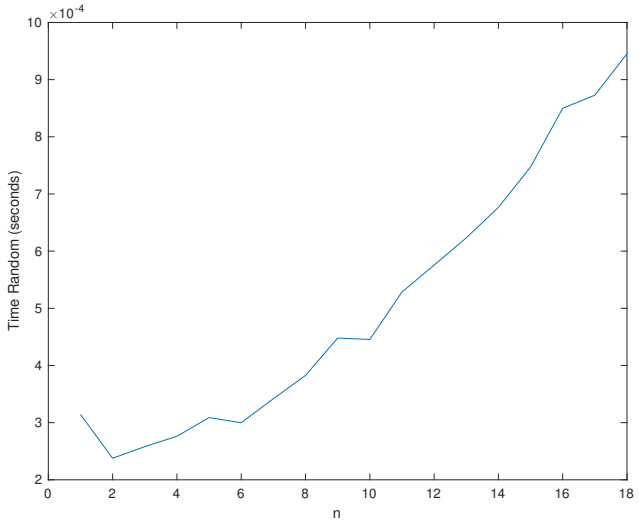

Figure 3: Vanlig simplex - slumptal

n	Klee och Minty (s)	Random (s)	Klee och Minty (i)	Random (i)
2	0.00023108	0.00020518	3	2.750
4	0.00032125	0.00028657	9	5.400
6	0.00074669	0.00033717	25	6.650
8	0.0019437	0.00045491	67	9.000
10	0.0052226	0.00055305	178	10.55
12	0.013051	0.00069321	466	12.70
14	0.034918	0.00084232	1219	15.50
16	0.094921	0.00010407	3193	16.75
18	0.25562	0.0011223	8361	23.30

Table 6: Blands Regel

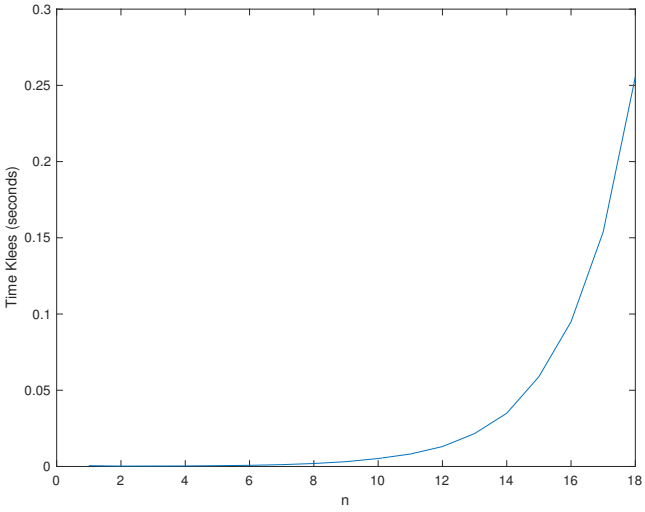

Figure 4: Blands - Klee och Minty.

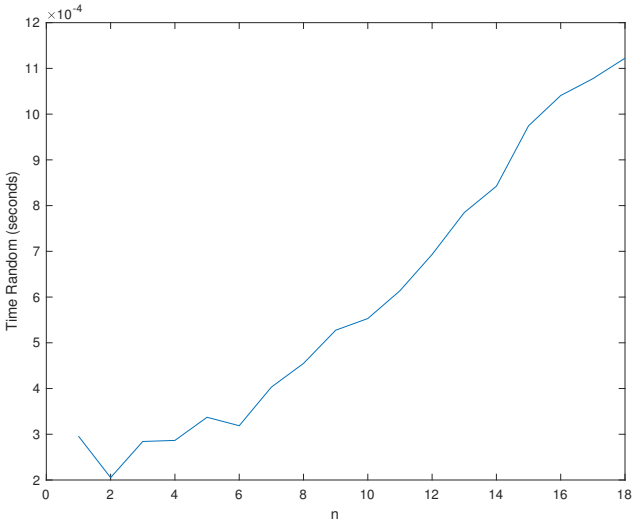

Figure 5: Blands - Slumptal.

n	Klee och Minty (s)	Random (s)	Klee och Minty (i)	Random (i)
2	2.9×10^{-3}	3.9×10^{-8}	1	1.3020
4	2.8×10^{-3}	5.0×10^{-8}	1	1.6860
6	3.2×10^{-3}	7.9×10^{-8}	1	2.0100
8	4.0×10^{-3}	9.9×10^{-8}	1	2.2990
10	4.7×10^{-3}	9.0×10^{-8}	1	2.4340
12	5.5×10^{-3}	1.7×10^{-7}	1	2.6320
14	6.4×10^{-3}	1.6×10^{-7}	1	2.8120
16	7.2×10^{-3}	1.9×10^{-7}	1	2.9640
18	8.2×10^{-3}	2.3×10^{-7}	1	3.0430

Table 7: Bästa målfunktionsvärde

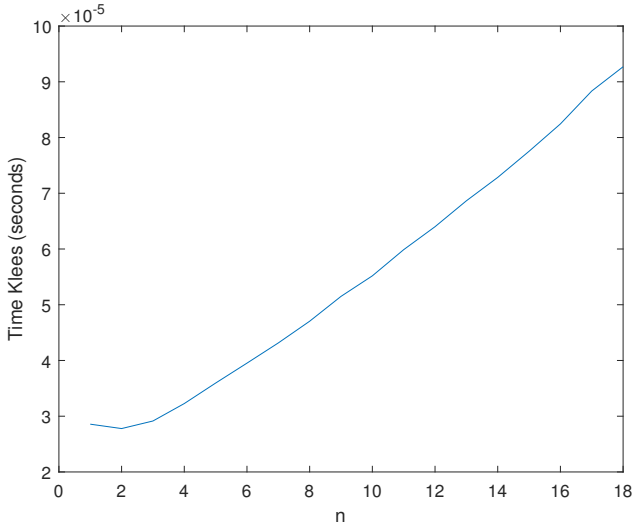

Figure 6: Bäst målfunktionsvärde - Klee och Minty

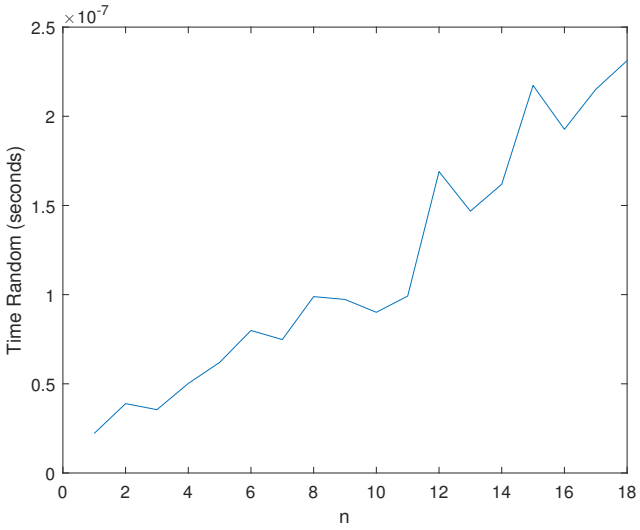

Figure 7: Bäst målfunktion - Slumptal

7 Diskussion

Ett genomgående tema för alla metoder är att de presterar snabbare vid körning med slumpade variabler kontra Klee och Mintys besvärliga familj. Den som dock utmärker sig mest är vanliga simplex. På den körning som visas i resultatet är skillnaden mer än 9 sekunder mellan de olika körningarna. Vidare presterade samtliga metoder liknande på samtliga körningar som vi utförde. Resultatet visar endast en specifik körning för vardera scenario men de var så pass konsekventa att vi drar en slutsats om vilken som presterade bäst av dem. Simplexmetoden vid körning på Klee och Minty som är implementerad med bästa målfunktionsvärde var den snabbaste, Blands regel näst snabbast och vanliga simplex långsammast.

Baserat på vårt resultat kan vi dra slutsatsen att den vanliga simplexmetoden med den besvärliga familjen växer exponentiellt både då vi ser till tid och iterationer. Vidare så växer metoderna som utgörs av Blands regel respektive Bästa målfunktionsvärdet av en polynomiell ökning vilket tydliggörs i våra plottar. Detta ses tydligast vad gäller jämförelse med Klee och Mintys besvärliga familjen som variabler, men samma mönster följs även vid körning av slumpade variabler från en likformigt diskret fördelning.

Det finns betydligt fler regler än de tre vi har använt i denna rapport som är bättre lämpade för vissa typer av problem. Som vi observerat i denna rapport kan valet av metoden medföra drastiska förändringar i dess effektivitet. Förslagsvis är en idé att man inför ett nytt problem undersöker hur olika regler presterar och utifrån testkörningarna välja ut den regel som är effektivast på just detta problem.

Något som vi har observerat under utförandet av laborationen är att den matematiska världen skiljer sig till hur vi implementerar i datorn. Eftersom vi i matematiken har reella tall med oändlig precision är matematik exakt, till kontrast till datoraritmetik. Om vi använder simplexmetoden i den matematiska världen och utför basbyte blir koefficienterna framför alla förutom basvariabeln till exakt noll. Vi observerar att detta inte sker i datorn utan att den ofta gör approximationer som gör att det istället är små tal mycket nära noll. Detta blir problematisk om man försöker göra denna till en basvariabel, då division på tal mycket nära noll gränsar mot oändligheten, som datorn inte heller kan beskriva till

skillnad från matematiken. Lösningen till detta som vi inte har implementerat i denna rapport, är att använda sig av toleransvärlden som särskiljer faktiskt noll och vad som är en approximation till noll i datorn. Man skulle exempelvis kunna tänka sig att tal under ett mycket lågt värde som exempelvis 10^{-8} hade uppfattats som att vara noll och ej utföra basbyten på dessa variabler med sådana små koefficienter.

Avslutningsvis, genom att analysera *Figure* 1 så kan vi tyda en problematik för den besvärliga familjen till följd av att metoden väljer att stegra igenom samtliga hörnpunkter, innan lösningen återfinns. Det innebär alltså att den optimal lösningen ges av den sista besökta hörnpunkten då vi använt oss av Klee och Mintys metod. Genom att undersöka *Figure* 1 så kan man understryka att det optimala hade varit att gå från $x = (0, 0)$ till $x = (0, 25)$ direkt för att hålla simplexmetoden effektivare. Med det sagt kan metoden orsaka komplikationer för dess effektivtet i takt med att n blir större.

8 Bibliography

References

[1] Lundgren Jan. “Optimeringslära”. In: vol. 3. Stundetlitteratur AB. 2013, pp. 80–90.

Appendix

8.1 Appendix Code - labb2_main.m

Källkod for `labb2_main.m`.

```
clear all; close all; clc

% Repeat means how many times to re-run the same to get a mean value
% of the amount of iterations and time. To get a cleaner plot.

% Iter means to how large value of n we wish to count to.

% Simplexmethod to choose: 'normal_simplex', 'blands_simplex',
% 'bestcost_simplex'

repeat = 100;
iter = 18;

simplex_method = 'bestcost_simplex';

time_klees = zeros(repeat,iter);
time_random = zeros(repeat,iter);
tot_iter_klees = zeros(repeat,iter);
tot_iter_random = zeros(repeat,iter);

for i = 1:repeat
    for n = 1:iter
        tic % start time
        [A,b,c] = generate_variables(n);

        if strcmp(simplex_method, 'normal_simplex')
            [maximum1, tableau, tot_iter] = mysimplex_example(A,b,c);
        elseif strcmp(simplex_method, 'blands_simplex')
            [maximum1, tableau, tot_iter] = mysimplex_blands(A,b,c);
        elseif strcmp(simplex_method, 'bestcost_simplex')
            [maximum1, tableau, tot_iter] = mysimplex_bestcost(A,b,c);
        else
```

```
            error('Simplex method not specified')
        end

        time_klees(i,n) = toc; % end time
        time_klees(1,n) = toc;

        tot_iter_klees(i,n) = tot_iter;
        tic
        [A,b,c] = generate_randomvariables(n);

        if strcmp(simplex_method, 'normal_simplex')
            [maximum1, tableau, tot_iter] = mysimplex_example(A,b,c);
        elseif strcmp(simplex_method, 'blands_simplex')
            [maximum1, tableau, tot_iter] = mysimplex_blands(A,b,c);
        elseif strcmp(simplex_method, 'bestcost_simplex')
            [maximum1, tableau, tot_iter] = mysimplex_bestcost(A,b,c);
        else
            error('Simplex method not specified.')
        end

        time_random(1,n) = toc;
        tot_iter_random(i,n) = tot_iter;
    end
end

figure(1)
plot(mean(time_klees,1));
xlabel('n');
ylabel('Time Klees (seconds)');

figure(2)
plot(mean(time_random,1));
xlabel('n');
ylabel('Time Random (seconds)')

figure(3)
plot(mean(tot_iter_klees,1));
```

```
xlabel('n');
ylabel('Total iterations (klee-minty constants)');

figure(4)
plot(mean(tot_iter_random,1));
xlabel('n');
ylabel('Total iterations (random constants)')
```

8.2 Appendix Code - generate_variables.m

Källkod for `generate_variables.m`.

```
function [A,b,c] = generate_variables(n)
%generate_variables
c = [];

for i = 1:n
    c(end+1) = 2^(n-i);
end

A = zeros(n,n);
b = zeros(n,1);

for j = 1:n
   for i = 1:j-1
      A(j,i) = 2 * 2^(j-i);
   end
   b(j,1) = 5^j;
   A(j,j) = 1;
end
end
```

8.3 Appendix Code - generate_randomvariables.m

Källkod for `generate_randomvariables.m`.

```
function [A,b,c] = generate_randomvariables(n)
%generate_variables
```

```
lowerb = 1;
upperb = 500;
c = [];

for i = 1:n
    c(end+1) = randi([lowerb, upperb]);
end

A = zeros(n,n);
b = zeros(n,1);

for j = 1:n
   for i = 1:j-1
      A(j,i) = randi([lowerb,upperb]);
   end
   b(j,1) = randi([lowerb,upperb]);
   A(j,j) = randi([lowerb,upperb]);
end
end
```

8.4 Appendix Code - mysimplex_example.m

Källkod for `mysimplex_example.m`.

```
function [maximum,tableau,tot_it] = mysimplex_example(A,b,c)

%mysimplex_example Simplex solver for LP problems, with steepest ascent.
    %Tries to solve the LP-problem
        %max z = c * x,
        %subject to A*x<=b,
        %such that x>=0, and b>=0.
    %Returns the maximum value of the objective function z.

tableau=zeros(size(A,1)+1,size(A,2)+size(A,1)+1);%Initialize a tableau,
                                                 %with room for slack.

tableau(1,1:size(A,2))=-c;                %Fill the tableau
```

```
tableau(2:end,1:size(A,2))=A;
tableau(2:end,end)=b;
tableau(2:end,size(A,2)+1:end-1)=eye(size(A,1)); %Add slack
t=1;
tot_it = 0;
while t == 1
    if size(find(tableau(1,1:end-1)<0))>0
        [~,ind]=min(tableau(1,1:end-1));       %Find incoming variable


        minimum=inf*ones(1,size(tableau(2:size(A,1)+1,ind),1));
        for j=2:size(A,1)+1
            if tableau(j,ind)>0                          %Find candidates
                minimum(j)=tableau(j,end)/tableau(j,ind);
            end                                        %for outgoing variable
        end
        [~, pivot]=min(minimum);                 %Find outgoing variable


        %Perform row operations on the tableau
        tableau(pivot,:)=tableau(pivot,:)./tableau(pivot,ind);

        index=1:size(A,1)+1;
        index=index(index~=pivot);
        for j=index
            tableau(j,:)=tableau(j,:)-tableau(j,ind)*tableau(pivot,:);
        end
    else
        t=0; %If no incoming variable found, we have the optimal value
    end
    tot_it = tot_it + 1;

end
maximum=tableau(1,end); %Optimal value
```

8.5 Appendix Code - mysimplex_blands.m

Källkod for `mysimplex_blands.m`.

```
function [maximum,tableau,tot_it] = mysimplex_blands(A,b,c)

%mysimplex_blands Simplex solver for LP problems, with blands rule.
    %Tries to solve the LP-problem
        %max z = c * x,
        %subject to A*x<=b,
        %such that x>=0, and b>=0.
    %Returns the maximum value of the objective function z.

tableau=zeros(size(A,1)+1,size(A,2)+size(A,1)+1);%Initialize a tableau,
                                                 %with room for slack.

tableau(1,1:size(A,2))=-c;                %Fill the tableau
tableau(2:end,1:size(A,2))=A;
tableau(2:end,end)=b;
tableau(2:end,size(A,2)+1:end-1)=eye(size(A,1)); %Add slack
t=1;
tot_it = 0;

while t == 1
    if size(find(tableau(1,1:end-1)<0))>0
        ind = find(tableau(1,1:end-1) < 0, 1); % Find incoming variable

        minimum=inf*ones(1,size(tableau(2:size(A,1)+1,ind),1));
        for j=2:size(A,1)+1
            if tableau(j,ind)>0                        %Find candidates
                minimum(j)=tableau(j,end)/tableau(j,ind);
            end                                      %for outgoing variable
        end
        [~, pivot]=min(minimum);               %Find outgoing variable


        %Perform row operations on the tableau
        tableau(pivot,:)=tableau(pivot,:)./tableau(pivot,ind);
```

```
        index=1:size(A,1)+1;
        index=index(index~=pivot);
        for j=index
            tableau(j,:)=tableau(j,:)-tableau(j,ind)*tableau(pivot,:);
        end
    else
        t=0; %If no incoming variable found, we have the optimal value
    end
    tot_it = tot_it + 1;
end
tot_it
maximum=tableau(1,end); %Optimal value
```

8.6 Appendix Code - mysimplex_bestcost.m

Källkod for `mysimplex_bestcost.m`.

```
function [maximum,tableau,tot_it] = mysimplex_bestcost(A,b,c)

%mysimplex_bestcost Simplex solver for LP problems, with best objective
% function.
    %Tries to solve the LP-problem
        %max z = c * x,
        %subject to A*x<=b,
        %such that x>=0, and b>=0.
    %Returns the maximum value of the objective function z.

tableau=zeros(size(A,1)+1,size(A,2)+size(A,1)+1);%Initialize a tableau,
                                                 %with room for slack.
tableau(1,1:size(A,2))=-c;                %Fill the tableau
tableau(2:end,1:size(A,2))=A;
tableau(2:end,end)=b;
tableau(2:end,size(A,2)+1:end-1)=eye(size(A,1)); %Add slack
t=1;
tot_it = 0;

while t == 1
```

```
if size(find(tableau(1,1:end-1)<0))>0
    best_score = -inf;
    best_pivot = -inf;
    best_ind = -inf;
    indices = find(tableau(1,1:end-1) < 0);

    % ind är kolumn negativa värden,
    % pivot

    for ind=indices
        minimum=inf*ones(1,size(tableau(2:size(A,1)+1,ind),1));

        for j=2:size(A,1)+1
            if tableau(j,ind)>0                    %Find candidates
                minimum(j)=tableau(j,end)/tableau(j,ind);
            end
        end

        [~, pivot] = min(minimum);

        %pivot är rad, ind är kolumnen
        score = tableau(1,end) - tableau(1, ind) .* ...
                (tableau(pivot, end) ./ tableau(pivot, ind));

        if score > best_score
            best_score = score;
            best_pivot = pivot;
            best_ind = ind;
        end
    end

    %Perform row operations on the tableau
    tableau(best_pivot,:)=tableau(best_pivot,:)./tableau(best_pivot,best_ind);
    index=1:size(A,1)+1;
    index=index(index~=best_pivot);

    for j=index
```

```
            tableau(j,:)=tableau(j,:)-tableau(j,best_ind)*tableau(best_pivot,:);
        end

        tot_it = tot_it + 1;

    else
        t=0; %If no incoming variable found, we have the optimal value
    end
end
maximum=tableau(1,end); %Optimal value
```

8.7 Appendix opponering

Introduktion:

1. Välformulerad. Ni skapar en bra röd tråd utifrån ursprungsfakta fram till hur ni ska presentera en lösning på laborationen.

Teori:

1. Kanske värt att nämna att den ”vanliga” simplexmetoden använder sig av ”steepest-ascent”.

2. Ni förklarar väldigt bra hur man tar sig mellan hörnpunkter genom basbyten, vilket är bra och ger läsarna en övergripande förståelse kopplat till de olika sökriktningarna ni nämner.

3. Fortfarande lite oklart när ni hänvisar till referenserna, om hyperlänken inte fungerar beroende på val av program för att öppna rapporten så tappar den lite sitt syfte och det blir oklart vad ”[1]” står för.

Analytisk lösning för n = 2:

1. Mycket bra figur (Figur 1)! När ni rödmarkerar hörnpunkterna i det tillåtna området så blir det väldigt intuitivt hur ni hittar hörnpunkterna i simplextablåerna var ni även använder er av rödmarkeringar.

2. Kan vara bra att använda er utav variabeln ”z” när ni definierar målfunktionen eftersom ni refererar eran teori till kurslitteraturen. Då får man en bättre bild i varför ”b-vektorn” i simplextablån blir 0 för målfunktionen.

3. I stycket ovanför Tabell 1, så skriver ni ”stegras” istället för ”stegas”.

4. Skriv kanske att radoperationer utförs innan ni skriver exempelvis ”(III) = (III) − 4(II)”.

5. I texten till Tabell 2 & framåt så skriver ni ”Ursprungliga simplextablån” igen. Är det verkligen den ursprungliga?

Utförande:

1. En sak vi funderade över var hur ni väljer att lägga upp rubrik 4 som ”Analytisk lösning för n = 2”. Det känns inte som att ni analyserar så mycket utan att hela den delen känns mer som ett ”Utförande” där ni visar hur ni löser Klee-Minty problemet teoretiskt och inte har så mycket att göra med just analysen för ”n = 2”. Huruvida så analyserar ni i sista stycket, men det skulle kanske även vara en bra punkt att ta upp i diskussionen istället?

2. På sida 11 så skriver ni i andra stycket (hänvisa till teorin). Har ni missat att skriva där?

3. Ni definierar variablerna ”i” och ”j”, men det kan vara bra att nämna vad det är lite kort. Exempelvis om det kanske är rader, kolumner eller så.

4. I sista stycket så kommer helt plötsligt ”steepest-ascent metoden” in från ingenstans.

Resultat:

1. I samtliga tabeller får ni decimaltal på iterationerna för de slumpade koefficienterna. Bör det vara så?

2. I Tabell 5, får ni 4 iterationer för Klee-Minty då n = 2. Ska det vara verkligen så, oberoende vilken punkt ni utgår ifrån? (Se eran analytiska lösning för n = 2).

3. Bra och tydliga grafer! Lätt att hänga med vilken metod dom hör till.

4. Iterationerna för den alternativa metoden som ger störst förbättring av målfunktionsvärdet med Klee-Minty får ni till väldigt höga värden. Ska det verkligen vara så höga värden eftersom man med den metoden hittar den totala förbättringen via den kortaste sökvägen. (Se eran analytiska lösning för n = 2).

Diskussion:

1. Ni kanske skulle kunna redovisa lite mer utförligt vad som händer med metoderna då ”n” blir större och större.

2. Det är bra att ni tar upp och diskuterar eventuella regler för att kunna förbättra simplexmetoden.

3. Intressant att ni tar upp i slutet om tolerans och hur det skulle kunna ge annorlunda värden vid tillämpning.

Övrigt:

En väldigt välskriven rapport överlag! Erat upplägg och innehåll gjorde det väldigt intressant att läsa igenom rapporten. Bra jobbat!