OPERATIONAL RESEARCH AND THE SIMPLEX METHOD

April 19, 2020

Benjamin Stuart Lowery
201700585

University of Hull
Department of Physics and Mathematics

Project supervisor: Dr. Richard Danyi

Abstract

From Supply chain management and finding better ways to manage stock, to financial institutions wanting to optimise profits; Linear Programming provides a pragmatic, straightforward and efficient way to turn real-world problems into a set of linear inequalities and equations. From a more general Operational Research standpoint, Linear Programming is an integral tool to help find optimal solutions to a range of problems within the area. By focusing on the most well known tool for optimising Linear Programming problems, namely the Simplex Method, we will discover how this is apparent from a practical and theoretical standpoint. Attaining a solid footing in what makes the method so adaptable to any $n-dimensional$ linear problem. We will also dive into areas like non-linear programming and Integer programming with Simplex, alongside a comparison to interior points methods, analysing how these other methods work to achieve similar results. Culminating in a final scrutiny of how the method holds up in the current world of fast computations, new innovations and a changing mathematical landscape.

Contents

Acknowledgements

I'd like to say thanks first and foremost to my supervisor, Dr. Richard Danyi, for providing advice throughout this project. I am also thankful to my course and house mates Connor, Sam and Ryan (x2) for useful comments and help throughout year. Alex, Fraser, Barry, Joe, and my parents provided motivation. Whilst a special thanks also goes to Georgia who kept me persevering, especially through the unique circumstances faced late on during the writing of this thesis.

1 Introduction

"It happened because during my first year at Berkeley I arrived late one day at one of Neyman's classes."

George B. Dantzig, a first year doctoral student at UC Berkeley, mistook a set of unsolved problems in statistics for a homework question. Scribbling them down and solving them over the next few days, Dantzig had found these problems slightly harder than a normal homework assignment. He threw his solutions onto the desk of his professor; expecting them to get lost within the clutter of Neyman's desk.

"About six weeks later, one Sunday morning about eight o'clock, [my wife] Anne and I were awakened by someone banging on our front door. It was Neyman." Dantzig recalled in a 1986 interview for the College Mathematics Journal [1] . What the young student had done, initially unbeknownst to him, was solve these statistical problems and had a giddy professor already writing his papers introduction to be sent for publication. From this, Dantzig begun a journey into mathematical stardom.

Eight years after this tale, forever ingrained into the minds of wannabe mathematicians, Dantzig was working as a mathematical advisor for the pentagon. Tasked by his department to computationally speed up logistical issues faced by the US Air Force, he developed techniques stemming from the infant field of Linear Programming to optimise said issues [2]. The method used was to be known as the Simplex method, but where does it come from and what even is Linear Programming?

The ideas of Linear Programming in the history of mathematics often starts with Dantzig's contributions, but its origins can be dated back to a few years earlier during World War II. Namely in the field of economics and with Soviet economist Leonid Kantorovic. In 1939, he developed the first forms of the Linear Programming problem for organising and planning production[3]. Cited as a founder of the field, Kantorovic's method revolved around finding dual variables and corresponding primal solution, linking how the results from one directly impact the other. We will see and define the idea of Primal and Dual later on in this paper, which will consist in a slightly adapted form than what Kantorovic designed. He would later go on to win the Stalin Prize[1] for his work in resource allocation stemming from ideas he developed in the operations research field.

Another important character to this field, amongst many others, was John Von Neumann. He was a proverbial rockstar in the mathematical world, dabbling in a variety of topics from quantum mechanics to pure mathematics to the early days of computer science and most importantly to us, Linear Programming. He contributed an important aspect to this field, Duality Theory, which we shall explore in **Section 3.4.1**. Involving the ideas of Primal and Dual Linear Programs, at this point it is not necessary to know what this involves and how we connect this field to our paper as a whole. However, it should be important to understand that this concept is pivotal to expanding and solving more complicated Linear Programs and highlighting a connection between optimisation by maximising or minimising.

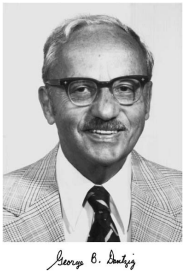
Knowing how key figures contributed to the field we are investigating, we can now consider Dantzig's key contribution to making this field what it is today. As mentioned earlier, the Simplex method was developed during Dantzig's time in the Pentagon. However the method was not instantly created as a perfect computational algorithm, it was still unable to traverse the difficulties Linear Programs could have in the methods initial conception. In fact, it wasn't until 1954-1955 that the implementation of the Simplex algorithm was improved and devloped enough

[1] Later known as the USSR State prize, awarded between 1941 and 1954 to people who had seen to further develop the Soviet Union

to handle 101 constraints[4]. Before this, the computers available had nowhere near the capabilities to cope with the implementation of the theory behind Simplex. It wasn't until IBM developed a computer "good enough" to successfully carry out complicated procedures. From this point, the increase in usability was more consistent and able to handle the complex Linear Programs it was intended to solve. With further developments throughout the latter half of the 21st century to improve performance.

(a) Leonid Kantorovich

(b) John von Neumann



(c) George B. Dantzig

Figure 1: Images (a) and (b) are in the public domain and taken from Wikimedia Commons. (c) is copyright of the National Academic Press.

Having explored some historical context and background it is worth mentioning that, overall, this paper is intended as a deep dive into the methods of the simplex algorithm and the modelling of real life scenarios into mathematical equations. Assuming basic knowledge of linear algebra and numerical analysis, other concepts, such as the simplex method, alternative optimization algorithms and duality theory will be introduced from scratch. Starting with an introduction to modelling Linear Programs and graphical representation of such problems, the simplex method for maximisation will then subsequently be introduced as well as more complicated examples and definition for a generalization of Linear Programming. Once a strong footing is attained, we will move onto caveats of the simplex method and what it means for problems to be infeasible and unbounded. Leading into minimisation problems and an excursion into duality theory and the connection between a maximisation and minimisation problem.

An important alternative to the simplex method, namely interior points methods, will be introduced in **Section 4**. Progressing to a look into a specific method within this classification. A conversation comparing these methods will ensue, analysing which is more useful for different scenarios. **Section 5** will compose of Integer programming and finding solutions to when we require integer solutions to optimisation queries. This section will also briefly look into non-linear programming, such as quadratic and higher programming and how this added complexity affects how we intend to use the simplex method to solve problems. The paper will finally make concluding remarks and the reader should be left with intrigue for this fascinating area of Mathematics.

2 Modelling Problems into Linear Programs

To get a better grasp of Linear Programming and the Simplex method, consider the following example:

Example 1. *Jon wants to invest some funds, £10,000 say, into the stock market. Not being experienced in the financial intricacies that this might entail, he decides to go through an investment firm in the hopes they can achieve profit more wisely than himself. Two companies, A and B offer their services. The former promises a 3% return on investment (ROI) after 2 years and limits new account holders to a maximum investment of £7000. Whereas the latter offers a more lucrative 5% after 2 years and no account limit. Furthermore, due to the apparent financial instability of Company B and general reliability of A, Jon decides to invest at least 3 times more into Company A than B. Finally, he does not necessarily want to invest all the money and is pursuing a maximisation in the ROI after 2 years.*

So how do we go about helping Jon and solving this problem? Our aim here is to try and apply Kantorovich's notion of attempting to formulate real life problems as a set of linear constraints and a maximisation function, then use Dantzig's Simplex algorithm to solve this. To achieve this, we can firstly introduce variables, x_1 and x_2 to denote how much we want to allocate into each fund; where these variables are in thousands of pounds. Then as stated in our problem, we want to maximise the return on investment. How we maximise this is dependent on a set of rules (or constraints) we want to stick by. To help with this, creating a table to layout the information given may be useful. We can see this,

	Company A (x_1)	Company B (x_2)	Restriction
ROI	3%	5%	None, we are maximising this.
Maximum investment	£7000	N/A	Need to invest at least 3× more in company A than B.

Utilising this information format, with the extra info of having a maximum of £10,000 to invest, we can now put this into algebraic inequalities modelled below:

$$\begin{aligned} x_1 &\leq 7 \\ x_1 &\geq 3x_2 \\ x_1 + x_2 &\leq 10 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Note, the last inequality ensure we cannot invest a negative amount, as it should be trivial to state this is absurd! If we then combine this with what we want to maximise and rearrange the constraints to make sure the LHS holds all the variables, we can now formulate the problem:

$$\begin{aligned} \text{maximise} \quad & 0.03x_1 + 0.05x_2 \\ \text{subject to} \quad & x_1 \leq 7 \\ & -x_1 + 3x_2 \leq 0 \\ & x_1 + x_2 \leq 10 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{2.1}$$

This now resembles a typical Linear Programming problem. As we can see, the ease of which we can take a wordy and perhaps daunting and confusing initial problem, then reduce it to a set of linear inequalities, highlights the attractiveness and usefulness of a technique such as this. The next step we need to ask ourselves is how we can use the Simplex method to find a solution to this problem.

2.1 Graphical Representation

To get a better grasp on how these inequalities affect each other, we can graph our constraints and create a *Feasible Region*. This is an area in which all possible solutions lie. On the boundary of this region will also be the Optimal Solution. The Optimal Value will lie on the corner points of the feasible region. With this in mind, we can see the Feasible Region in the following figure:

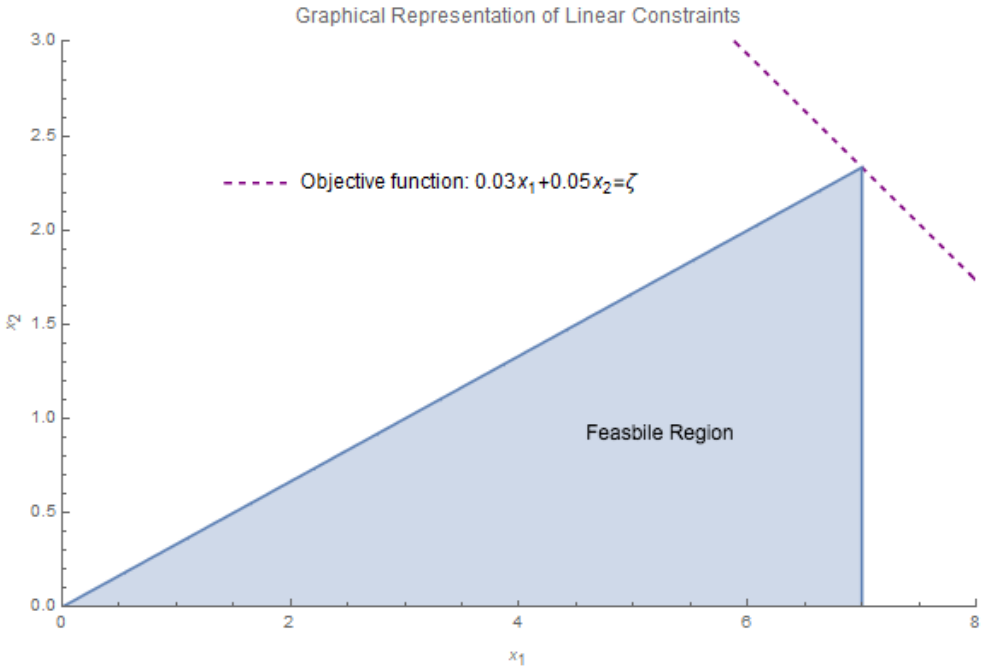


Figure 2: The above highlights what a typical feasible and bounded problem would look like. The Objective Function is currently located on the optimal value.

Here, the feasible region takes the form of a triangle, giving very few corner points. Finding an optimal solution in this case from a graphical point of view, is a fairly trivial task. Our corner points being attained at the following vertices of the triangle:

(x_1, x_2)	Optimum Value
$(7, 2.33)$	0.327
$(0, 0)$	0
$(7, 0)$	0.21

It can then be concluded the optimal solution lies at $x_1 = 7,\ x_2 = 2.33$ with an optimum value of 0.327, which was calculated through substituting into the maximisation function. We will discuss what this means after utilising the Simplex method to solve this problem in an alternative way.

2.2 Solving using Simplex method

What we want to do first is convert these inequalities and our maximisation equation (which will henceforth be noted as an Objective Function) into a set of equations under the same form. The *Objective Function*, we can simply set this equal to an arbitrary variable which we will denote as ζ. The Objective Function will then be negated to create an equation in the form:

$$\zeta - \textit{Objective Function} = 0$$

Whilst our inequalities can be converted to equalities with the addition of a weight w. These weights are known as slack variables in Linear Programming notation. Henceforth, we have the following set of equations:

$$\begin{aligned} \zeta - 0.03x_1 - 0.05x_2 \quad &= 0 \\ x_1 \quad + w_1 &= 7 \\ -x_1 + 3x_2 \quad + w_2 &= 0 \\ x_1 + x_2 \quad + w_3 &= 10 \end{aligned}$$

We now essentially have a set of linear equations to solve in which we can now apply the Simplex method to. There are multiple ways to structure the Simplex method such as a dictionary or as a tableau. For this paper and consistency with most written work on the subject, we will use the tableau method. With this, we can now express our set of equations in the following format:

ζ	x_1	x_2	w_1	w_2	w_3	Value
1	-0.03	-0.05	0	0	0	0
0	1	0	1	0	0	7
0	-1	3	0	1	0	0
0	1	1	0	0	1	10

Table 1: The Initial Simplex Tableau

Now it is worth asking what steps we can take to bring this tableau into an optimised form? The Simplex Method has a few steps to achieve this which culminates in the following procedure:

Simplex Method

1. Select column with the greatest negative number coefficient of the Objective Function. This is the pivot column.
2. For every item in the pivot column (excluding the first objective row) divide the row's corresponding *value* by said item. Ignoring any negative or possible dividing by 0.
3. Locate the smallest non-negative value, the item in the pivot column is what we will use as our *pivot*.
4. Carry out row operations to make all items in the column 0 (except the *pivot*).
5. Finally divide the pivot row by the pivot value.
6. Repeat.
7. Optimal solution is attained when no negatives are present in the Objective Function Row.

Once this process is completed, we then correspond the numbers to the pivot and we will have our optimal solution ζ. Alongside this are corresponding x_1, x_2 values. It should be noted that although we attain values for w_1, w_2, w_3, we are not concerned with the values these hold as these are *dummy variables*. Using Table 1 as the initial tableau, the Simplex Method can now be carried out below with each iteration, pivot calculation for the current iteration and the row operations required to progress to the next stage are highlighted below.

ζ	x_1	x_2	w_1	w_2	w_3	Value	Pivot Calculation	Required Row Operations
Initial Tableau								
1	-3/100	-1/20	0	0	0	0	-	$R_1 \rightarrow R_1 + \frac{1}{60}R_3$
0	1	0	1	0	0	7	-	$R_2 \rightarrow R_2$
0	-1	(3)	0	1	0	0	$\rightarrow \frac{0}{3} = 0$	$R_3 \rightarrow \frac{1}{3}R_3$
0	1	1	0	0	1	10	$\frac{10}{1} = 10$	$R_4 \rightarrow R_4 - \frac{1}{3}R_3$
Iteration 1								
1	-7/150	0	0	1/60	0	0	-	$R_1 \rightarrow R_1 + \frac{7}{150}R_2$
0	(1)	0	1	0	0	7	$\rightarrow \frac{7}{1} = 7$	$R_2 \rightarrow R_2$
0	-1/3	1	0	1/3	0	0	-	$R_3 \rightarrow R_3 + \frac{1}{3}R_2$
0	4/3	0	0	-1/3	1	10	$\frac{10}{4/3} = \frac{15}{2}$	$R_4 \rightarrow R_4 - \frac{4}{3}R_2$
Iteration 2								
1	0	0	7/150	1/60	0	49/150	-	-
0	1	0	1	0	0	7	-	-
0	0	1	1/3	1/3	0	7/3	-	-
0	0	0	-4/3	-1/3	1	2/3	-	-

As is evident, all options have been exhausted since the Objective Function row is all non-negative. Thus, we can create a solution set out of this optimised final iteration. To do so, we go through each column and if the column

contains zeros in all but one row, then we correspond the value on the row which has a non zero value to this variable. If we have more than one non-zero number in a column, this is known as a *free variable* and we set this to 0. I.e. from this explanation, we get the solution set of

$$(x_1, x_2, w_1, w_2, w_3) = \left(7, \frac{7}{3}, 0, 0, \frac{2}{3}\right)$$

$$\zeta = \frac{49}{150} \approx 0.327.$$

Hopefully, it is clear where these numbers come from. Moving on, these solutions demonstrate that, according to the parameters we set, an investment of £7000 in Company A and £2330 in Company B is advised. Additionally, the Optimal Value ζ indicates an increase of 32.7% will be expected on the portfolio after 2 years.

3 Higher dimensional and other Linear programs

3.1 Three Dimensions

Consider this new example building on the one from the previous section

Example 2. *Suppose Jon has an extra company to invest in, Company C, offering their investment services. Company C offers a 4% return on investment after a 2 year period; with a maximum initial investment of £4000 and a reputation for the Company to be considered reliable. If we were to incorporate the conditions from* **Example 1**, *we want to additionally state that we wish to invest at least 3 times more in company A and C than B. Finally, Jon now has a cash balance of £15,000 to invest.*

We can begin to solve this by adapting (2.1) and altering the set of equations using information from the above example. This yields the following Linear Program:

$$\begin{aligned} \text{maximise} \quad & 0.03x_1 + 0.05x_2 + 0.04x_3 \\ \text{subject to} \quad & x_1 \leq 7 \\ & x_3 \leq 4 \\ & -(x_1 + x_3) + 3x_2 \leq 0 \\ & x_1 + x_2 + x_3 \leq 15 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

From a Simplex point of view, this can be solved in the same process as other problems we saw with two variables and we will see later how this is. However, when three variables are in place, these problems begin to get interesting when looking and analysing the graphical representation of said problems. If we plot these constraints on a 3D axis and create a Feasible Region, we arrive at the following:

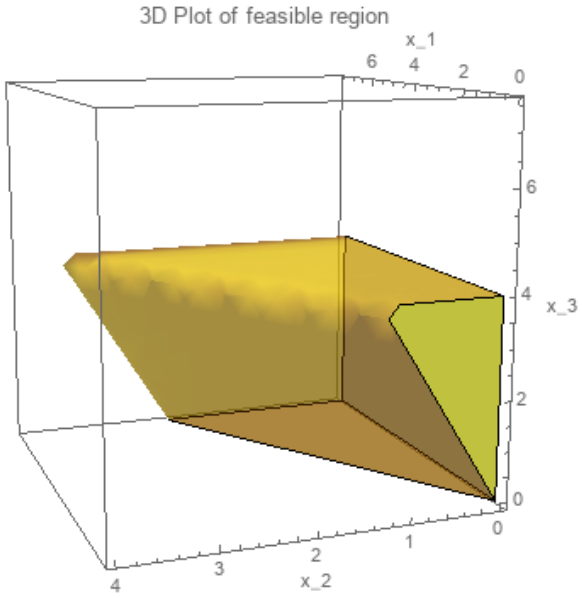


Figure 3: The feasible region for our three variable Linear Program defined in **Example 2**. The odd shape highlights the difficulty of solving three dimensional problems from a purely graphical viewpoint.

As we can see, in comparison to the graphs for two variables, it is harder to initially see where this maximum will lie (Well, maybe an educated guess could be made). We could use a graphical method to solve this problem.[2] However for brevity, we will take a different approach to that has been done so far, and use the Simplex Method and graphical representation in tandem. This is achieved by seeing how each iteration of the algorithm improves our guess to find the maximum and allow a visualization of the progression. Thus, aiming to create a more coherent understanding on how the Simplex Method improves to an Optimal Solution. We can begin by first creating our Tableau from the constraints; as previously achieved with two variables. We can first adjust our constraints into a set of equalities with the introduction of Slack (weight) variables. Thus,

$$\begin{aligned}
\zeta - 0.03x_1 - 0.05x_2 - 0.04x_3 &= 0 \\
x_1 + w_1 &= 7 \\
x_3 + w_2 &= 4 \\
-x_1 + 3x_2 - x_3 + w_3 &= 0 \\
x_1 + x_2 + x_3 + w_4 &= 15
\end{aligned}$$

are our equations which can then be formulated and have the Simplex Method applied to them. Giving the following tableau:

[2] See [5] for an insight to how this could be achieved.

ζ	x_1	x_2	x_3	w_1	w_2	w_3	w_4	Value	Pivot Calculations	Required Row operations
Initial Tableau										
1	-3/100	-1/20	-1/25	0	0	0	0	0	-	$R_1 \to R_1 + \frac{1}{60}R_4$
0	1	0	0	1	0	0	0	7	-	$R_2 \to R_2$
0	0	0	1	0	1	0	0	4	-	$R_3 \to R_3$
0	-1	(3)	-1	0	0	1	0	0	$\to \frac{0}{3} = 3$	$R_4 \to \frac{1}{3}R_4$
0	1	1	1	0	0	0	1	15	$\frac{15}{1} = 15$	$R_5 \to R_5 - \frac{1}{3}R_4$
Iteration 1										
1	-7/150	0	-17/300	0	0	1/60	0	0	-	$R_1 \to R_1 + \frac{17}{300}R_3$
0	1	0	0	1	0	0	0	7	-	$R_2 \to R_2$
0	0	0	(1)	0	1	0	0	4	$\to \frac{4}{1} = 4$	$R_3 \to R_3$
0	-1/3	1	-1/3	0	0	1/3	0	0	-	$R_4 \to R_4 + \frac{1}{3}R_3$
0	4/3	0	4/3	0	0	-1/3	1	15	$\frac{15}{4/3} = 45/4$	$R_5 \to R_5 - \frac{4}{3}R_3$
Iteration 2										
1	-7/150	0	0	0	17/300	1/60	0	17/75	-	$R_1 \to R_1 + \frac{7}{150}R_2$
0	(1)	0	0	1	0	0	0	7	$\to \frac{7}{1} = 7$	$R_2 \to R_2$
0	0	0	1	0	1	0	0	4	-	$R_3 \to R_3$
0	-1/3	1	0	0	1/3	1/3	0	4/3	-	$R_4 \to R_4 + \frac{1}{3}R_2$
0	4/3	0	0	0	-4/3	-1/3	1	29/3	$\frac{29/3}{4/3} = \frac{29}{4}$	$R_5 \to R_5 - \frac{4}{3}R_2$
Iteration 3										
1	0	0	0	7/150	17/300	1/60	0	83/150	-	-
0	1	0	0	1	0	0	0	7	-	-
0	0	0	1	0	1	0	0	4	-	-
0	0	1	0	1/3	1/3	1/3	0	11/3	-	-
0	0	0	0	-4/3	-4/3	-1/3	1	1/3	-	-

Here, we now have our optimal solution set as

$$(x_1, x_2, x_3, w_1, w_2, w_3, w_4) = \left(7, \frac{11}{3}, 4, 0, 0, 0, \frac{1}{3}\right)$$
$$\zeta = \frac{83}{150} \approx 0.553.$$

From this we know that we need to give £7,000, £3,666 and £4,000 to companies A,B and C respectively. With a yield of 55.3% return for our initial investment after 2 years. We can asses this result as, in comparison to using just two companies, the addition of the third which offers a middle of the road package between companies A and B, with a greater % return than what would be expected with just two companies. This is with the addition of a larger pool of funds, which could have potentially watered down our relative expected increase in funds. All in all, Linear Programming is a useful and practical tool for Portfolio Management, and not just a trivial theoretical experiment.

Although, in a real world environment, companies will not solely rely on modelling a Linear Program to dictate how to invest (as there are concepts and context which cannot be expressed solely as a linear equation). We will discuss in a later Section ways in which we can solve investment problems with added complexity.

If we now refer back to our comments earlier on visualising the Simplex Method on our plot. We can use *Mathematica* to plot the points for our optimal solution after each iteration. These being:

$$\underline{\text{Iteration 1:}}\ (x_1, x_2, x_3) = (0,0,0)\ \ \zeta = 0$$
$$\underline{\text{Iteration 2:}}\ (x_1, x_2, x_3) = (0,4/3,4)\ \ \zeta = \frac{17}{75}$$
$$\underline{\text{Iteration 3:}}\ (x_1, x_2, x_3) = (7,11/3,4)\ \ \zeta = \frac{83}{150}$$

and in doing so we get the progression of our method as follows:

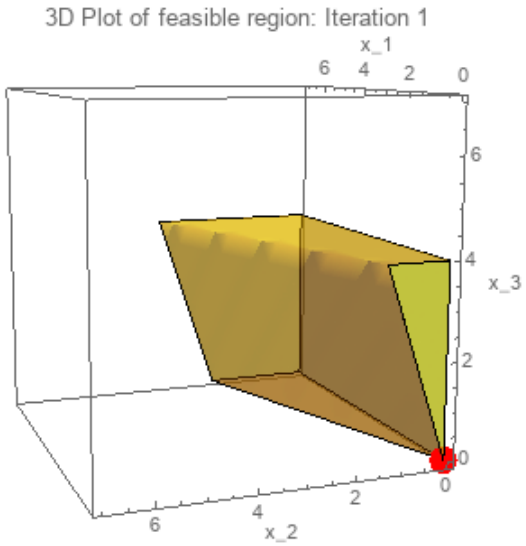


Figure 4: Initial Tableau/Iteration 1 with co-ordinates (0,0,0).

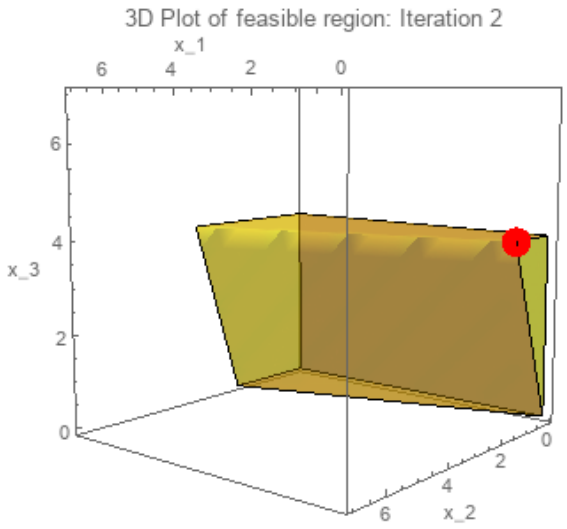


Figure 5: Iteration 2 with co-ordinates (0,4/3,4).

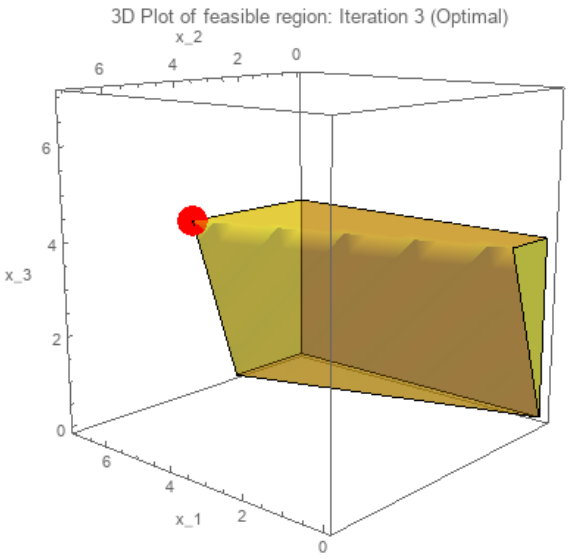


Figure 6: Iteration 3 an Optimal Solution with co-ordinates (7,11/3,4).

This is a very useful way to see how fast the Simplex Method is able to tend to our optimal points in a matter of a few iterations. Although, as it should be no surprise, three variables is our limit to create these visual representations of the method. However, the Simplex Method can carry on for an arbitrary number of variables. So how can we now generalise the method to n variables from a now completely mathematical point of view?

3.2 Higher Dimensions

At this point it is likely worthwhile to now generalise our ideas to express problems of n-dimensions. Consider firstly what it means for something to be in canonical form; that is, we can express problems with constraints exclusively with "≤" inequalities in the following definition.

Definition 3.1. *A Linear Program can be considered to be in Canonical Form if it is structured as:*

$$\begin{aligned} \text{Maximise} \quad & \zeta = c^T \mathbf{x} \\ \text{Subject to:} \quad & A\mathbf{x} \leq \mathbf{b} \\ & x_{1,..,n} \geq 0 \end{aligned}$$

We can also define what is a Standard Form of a Linear Program; that is one in which we substitute the inequalities with equalities by introducing weight variables to accommodate this change. This was seen in previous examples to be the step in which we converted our problems into a form that could then be expressed in tableau format. So, generally speaking we can now define a Standard Form.

Definition 3.2. *A Linear Program is considered to be in Standard Form if it takes the form:*

$$\begin{aligned} \text{Maximise} \quad & \zeta - c^T \mathbf{x} = 0 \\ \text{Subject to:} \quad & A\mathbf{x} + \mathrm{I}\mathbf{w} = \mathbf{b} \\ & x_{1,..,n}, w_{1,..,m} \geq 0 \end{aligned}$$

[6]

Currently we have assumed that only "≤" constraints are possible. If one was to have a constraint with alternative inequalities, we need to introduce different weight variables when converting to Standard Form. Hence, consider the following different permutations a constraint can attain:

$$\sum_{j=1}^{n} a_{ij} x_j \left\{ \begin{array}{l} \leq \\ = \\ \geq \end{array} \right\} b_i \quad (i = 1, ..., m).$$

From this, the first inequality, as we already know, is replaced by a slack variable (w). Then, for the equality constraints we add an artificial variable (a); whereas "≥" is accommodated the negation of a slack variable and

addition of an artificial variable. In mathematical terms this is more succinctly expressed as:

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1m}x_m \leq b_1 \implies a_{11}x_1 + a_{12}x_2 + ... + a_{1m}x_m + w_1 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2m}x_m = b_2 \implies a_{21}x_1 + a_{22}x_2 + ... + a_{2m}x_m + \alpha_1 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + ... + a_{3m}x_m \geq b_3 \implies a_{31}x_1 + a_{32}x_2 + ... + a_{3m}x_m - w_2 + \alpha_2 = b_3$$
$$a_1, a_2, w_1, w_2 \geq 0.$$

[6]

It should be noted that what we denote these weights can differ depending on the reading materials and an authors conventions.

A final generalised point we have seen but not defined, is the name we give to a solution at each iteration. It has been noted in 2-D and 3-D that our solutions iterate through vertices and points in the feasible region. Despite not being able to visualise higher dimensions, we still have these solution sets we are iterating through. These are feasible, but not necessarily optimal, and satisfy our Linear Program. These are known as **Basic Feasible Solutions**. We can more formally define this below in a perhaps more complicated and highbrow manner than what is required to understand the concept. Yet this is a concrete definition in which we build intuitive understandings from.

Definition 3.3. *(Basic Feasible Solution)*
Consider a Linear Program in standard form (3.2), where **W** *can contain any type of weight variable. A Basic Feasible Solution is a feasible solution* $x \in \mathbb{R}^n$ *for which there exists a basis,* B*, of* m*-elements i.e.*

$$B \subseteq \{1, 2, ..., n\}$$

And satisfies the following parameters:

- *The square matrix* A_B *is non-singular*
- $x_j = 0, \forall\, j \notin B$.

[7]

3.3 Special Cases

3.3.1 Infeasible

Thinking about Linear Programming in a graphical sense, an infeasible Linear Program is one in which no feasible region can be found. If we look at the following Linear Program:

$$\begin{aligned} \text{maximise} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 + 3x_2 \geq 13 && (3.1) \\ & 3x_1 + 2x_2 \leq 12 && (3.2) \\ & x_1 + 2x_2 \leq 8 && (3.3) \\ & x_1, x_2 \geq 0 \end{aligned}$$

We can solve why this Linear Program can't work by taking the first and third constraint and see where they cross. I.e.

$$\begin{aligned} x_1 + 2x_2 = 8 \Rightarrow \quad x_1 &= 8 - 2x_2 \\ x_1 + 3x_2 = 13 \Rightarrow \quad x_1 &= 13 - 3x_2 \\ \Rightarrow 13 - 3x_2 &= 8 - 2x_2 \\ \Rightarrow x_1 = -2, x_2 &= 5. \end{aligned}$$

We can see this crosses in the negative x-axis, which contradicts the $x_1, x_2 \geq 0$ constraint. If we want to visualise this graphically, we can plot the constraints and in the following figure 7, see how the Linear Program cannot be solved:

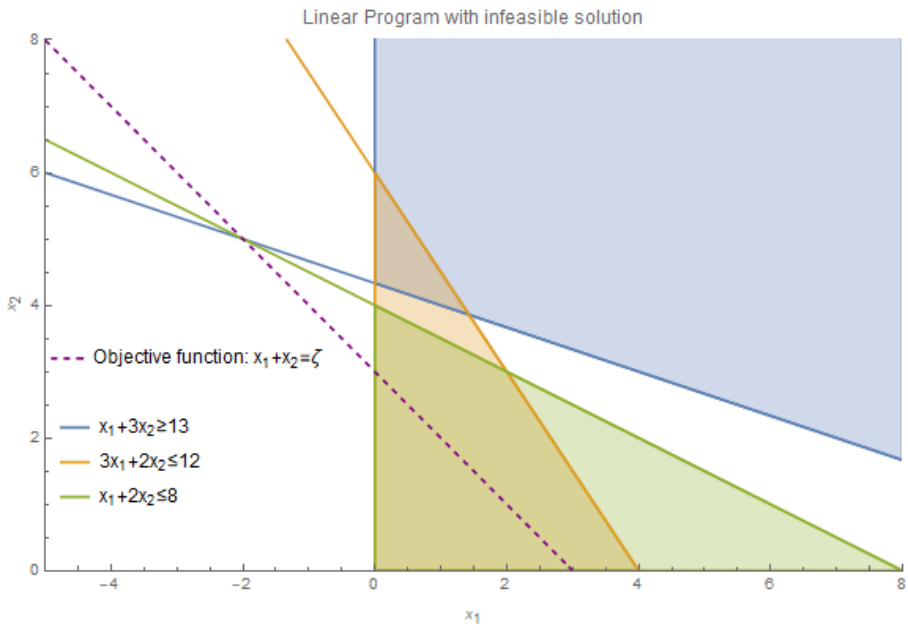


Figure 7: The above highlights what a typical infeasible and bounded problem would look like, as we can see there is no feasible region. The Objective Function is currently set at $\zeta = 5$ but we can see it only intersect two constraints at any point on this line

Here it can more clearly be seen that there is no section in which three constraints intersects with each other. Alternatively, we can carry out the Simplex Method to see why this won't work. For an infeasible solution, it must be so that our "supposed" optimal solution when we exhaust all options, does not satisfy all the constraints. The tableau for this Linear Program problem follows a similar tableau as previous example, with the weight variables added stemming from our generalisation in the previous section. We then have the following equations:

$$\begin{aligned} \zeta - x_1 - x_2 &= 0 \\ x_1 + 3x_2 - w_1 + a_1 &= 13 \\ 3x_1 + 2x_2 + w_2 &= 12 \\ x_1 + 2x_2 + w_2 &= 8 \end{aligned}$$

As will eventually be seen in solving the following tableau, we will have two options for picking our pivot column. Although it doesn't matter which one we pick, we should consider following a consistent process. Hence, in this example and further examples, if two pivot column options will exist, the rightmost option will be chosen.

ζ	x_1	x_2	w_1	w_2	w_3	a_1	Value	Pivot Calculations	Required Row operations
Initial Tableau									
1	-1	-1	0	0	0	0	0	-	$R_1 \rightarrow R_1 + \frac{1}{2}R_4$
0	1	3	-1	0	0	1	13	$\frac{13}{3} = 4.33$	$R_2 \rightarrow R_2 - \frac{3}{2}R_4$
0	3	2	0	1	0	0	12	$\frac{12}{2} = 6$	$R_3 \rightarrow R_3 - R_4$
0	1	(1)	0	0	1	0	8	$\rightarrow \frac{8}{2} = 4$	$R_4 \rightarrow \frac{1}{2}R_4$
Iteration 1									
1	1/2	0	0	0	1/2	0	4	-	-
0	-1/2	0	-1	0	-3/2	1	1	-	-
0	2	0	0	1	-1	0	4	-	-
0	1/2	1	0	0	1/2	0	4	-	-

As we can see, we cannot go any further than this point as we've exhausted all negative values in the Objective Function. We thus have the supposed optimal solution of $x_1 = 0, x_2 = 4, \zeta = 4$. However, when placing this into constraint (3.1), we have that

$$0 + 3(4) \ngeq 13.$$

Hence, an infeasible solution.

3.3.2 Unbounded

An Unbounded Program is one in which the feasible region can go to infinity (for a maximisation problem). Conversely, a minimisation problem which is unbounded will have an arbitrarily small value that can never be found. Like previously, consider this maximisation Linear Programming problem:

$$\begin{aligned} \text{maximise} \quad & 10x_1 + x_2 \\ \text{subject to} \quad & -2x_1 + 5x_2 \leq 5 \\ & 4x_1 + 6x_2 \geq 8 \\ & x_1 + x_2 \geq 0.5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

from these constraints, there will be a feasible region that is bounded by the second and third constraints heading positively towards infinity. With the first constraint, although bounding a feasible region from above, it is a constantly increasing line. Thus, if we was to plot these constraints we would get the following graph:

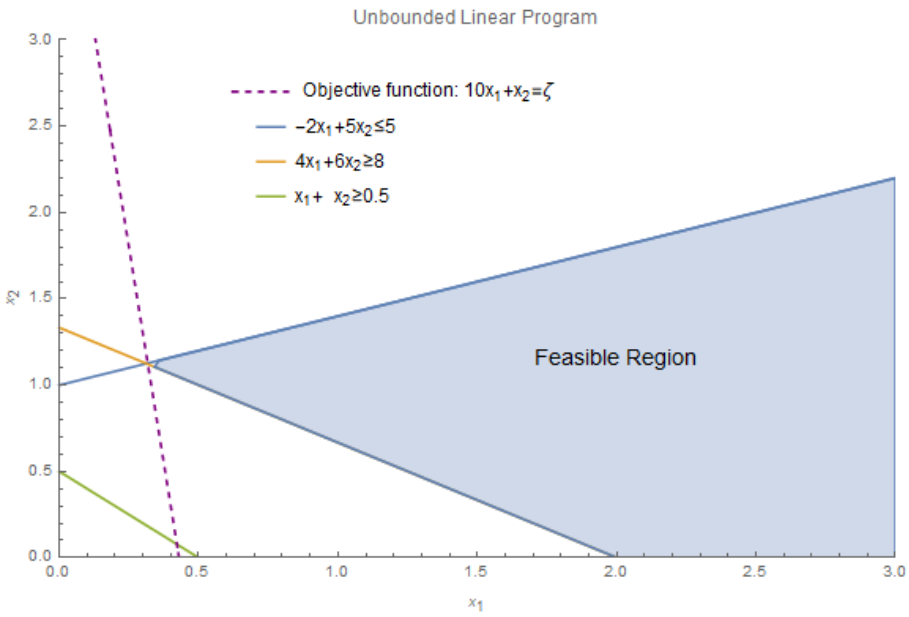


Figure 8: The above shows a unbounded Linear Program, if we were to maximise this problem, we can see that the feasible region carries on to infinity and a solution will not be found no matter how we move the Objective Function. Ergo, the region is unbounded.

We can clearly see that this feasible region will never end. Therefore an optimal solution will not be found since we could always find a larger point from any we choose.

If one was to solve this from a Simplex Method point of view, akin to our infeasibility Linear Program, we have artificial variables alongside slack variables due to the presence of "≥" constraints. Resulting in the following set of equations:

$$\begin{aligned} \zeta - 10x_1 - x_2 &= 0 \\ -2x_1 + 5x_2 + w_1 &= 5 \\ 4x_1 + 6x_2 - w_2 + a_1 &= 8 \\ x_1 + x_2 - w_3 + a_2 &= 0.5 \end{aligned}$$

Before we delve into the tableau and Simplex Method, we should note a slight tweak we are making to the Simplex Method to deal with the mixed constraints present. In this unique case, we will choose the **smallest** negative number coefficient as our pivot[3]. In addition to this, another quirk we have that affects all cases is that of if our pivot column cannot contain a pivot item, then we can move to the next negative number along the Objective Function row that satisfies the situational condition. With all this in mind, we can see the following iterations and Simplex tableau below:

[3] Another way to solve this is using Big-M method, this is beyond the realm of interest in this paper.

ζ	x_1	x_2	w_1	w_2	w_3	a_1	a_2	Value	Pivot Calculations	Required Row Operations
Initial Tableau										
1	-10	-1	0	0	0	0	0	0	-	$R_1 \rightarrow R_1 + R_4$
0	-2	5	1	0	0	0	0	5	$\frac{5}{5} = 1$	$R_2 \rightarrow R_2 - 5R_4$
0	4	6	0	-1	0	1	0	8	$\frac{8}{6} = 1.33$	$R_3 \rightarrow R_3 - 6R_4$
0	1	(1)	0	0	-1	0	1	1/2	$\rightarrow \frac{1/2}{1} = 0.5$	$R_4 \rightarrow R_4$
Iteration 1										
1	-9	0	0	0	-1	0	1	1/2	-	$R_1 \rightarrow R_1 + \frac{1}{5}R_2$
0	-7	0	1	0	(5)	0	-5	5/2	$\rightarrow \frac{5/2}{5} = 0.5$	$R_2 \rightarrow \frac{1}{5}R_2$
0	-2	0	0	-1	6	1	-6	5	$\frac{5}{6} = 0.833$	$R_3 \rightarrow R_3 - \frac{6}{5}R_2$
0	1	1	0	0	-1	0	1	1/2	-	$R_4 \rightarrow R_4 + \frac{1}{5}R_2$
Iteration 2										
1	-52/3	0	1/5	0	0	0	0	1	-	$R_1 \rightarrow R_1 + \frac{13}{8}$
0	-7/5	0	1/5	0	1	0	-1	1/2	-	$R_2 \rightarrow R_2 + \frac{7}{32}R_3$
0	(32/5)	0	-6/5	-1	0	1	0	2	$\rightarrow \frac{2}{32/5}$=0.3125	$R_3 \rightarrow \frac{5}{32}R_3$
0	-2/5	1	1/5	0	0	0	0	1	-	$R_4 \rightarrow R_4 + \frac{1}{16}R_3$
Iteration 3										
1	0	0	-7/4	-13/8	0	13/8	0	17/4	-	$R_1 \rightarrow R_1 + 14R_4$
0	0	0	-1/16	-7/32	1	7/32	-1	15/16	-	$R_2 \rightarrow R_2 + \frac{1}{2}R_4$
0	1	0	-3/16	-5/32	0	5/32	0	5/16	-	$R_3 \rightarrow R_3 + \frac{3}{2}$
0	0	1	(1/8)	-1/16	0	1/16	0	9/8	$\rightarrow \frac{9/8}{1/8} = 9$	$R_4 \rightarrow 8R_4$
Iteration 4										
1	0	14	0	-5/2	0	5/2	0	20	-	-
0	0	1/2	0	-1/4	1	8/32	-1	3/2	-	-
0	1	3/2	0	-1/4	0	1/4	0	2	-	-
0	0	8	1	-1/2	0	1/2	0	9	-	-

In this iteration, we can see that w_2 is our pivot column, yet each item in that column is negative thus a pivot cannot be chosen for the next iteration. Yet we still have negatives in the row of the Objective Function indicating we have not truly finished the Simplex Method. Under these conditions we say the Linear Program is unbounded and thus this example cannot be solved for a maximisation problem.

3.4 Minimisation

A major part of Linear Programming and the Simplex Method is a minimisation problem. This is the case when, instead of considering a maximum optimal value, we wish to instead choose a value in which we attain the smallest feasible solution. The best way to show how we solve a minimisation problem is by a simple example.

Example 3. Consider the unbounded problem we used in the previous subsection, i.e.

$$\begin{aligned} \text{maximise} \quad & 10x_1 + x_2 \\ \text{subject to} \quad & -2x_1 + 5x_2 \leq 5 \\ & 4x_1 + 6x_2 \geq 8 \\ & x_1 + x_2 \geq 0.5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

If, instead of maximising we minimise the objective, we can instead write the Linear Program as:

$$\begin{aligned} \text{minimise} \quad & 10x_1 + x_2 \\ \text{subject to} \quad & -2x_1 + 5x_2 \leq 5 \\ & 4x_1 + 6x_2 \geq 8 \\ & x_1 + x_2 \geq 0.5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

So how can we solve this? We know how to solve maximisation problems so it would be handy if there was a way to convert our minimisation problem into a form that we are used to solving. This can be done by introducing the Dual Simplex Method. Taken from the area of Duality theory which will be explored in the next section, we can carry out a set of steps to convert the Linear Program from minimisation to maximisation; this then leads to something we can solve easily. We can begin our solution to conversion by firstly converting all "≤" to "≥" equalities and then creating a matrix of coefficients from these constraints and Objective Function and transposing this. Thus,

$$\begin{aligned} -2x_1 + 5x_2 &\leq 5 \\ 4x_1 + 6x_2 &\geq 8 \\ x_1 + x_2 &\geq 0.5 \end{aligned} \qquad \Longrightarrow \qquad \begin{aligned} 2x_1 - 5x_2 &\geq -5 \\ 4x_1 + 6x_2 &\geq 8 \\ x_1 + x_2 &\geq 0.5 \end{aligned}$$

and,

$$\begin{pmatrix} 10 & 1 & \zeta \\ 2 & -5 & -5 \\ 4 & 6 & 8 \\ 1 & 1 & 0.5 \end{pmatrix}^T \Longrightarrow \begin{pmatrix} 10 & 2 & 4 & 1 \\ 1 & -5 & 6 & 1 \\ \zeta & -5 & 8 & 0.5 \end{pmatrix}.$$

This can subsequently be converted to a new set of constraints and an Objective Function. Since we have taken the transpose, we now have effectively taken a change of variable (this can be formally defined in due course). So now it is required to equate these values to a new set of variables, $y_{1,2,3}$ say. Finally, by taking our left hand column to be our new right hand side values, with the columns 2,3 and 4 corresponding as coefficients to our new variable base. This leads to the following Linear Program in which we now have a maximisation problem with purely "≤" constraints.

$$\begin{aligned} \text{maximise} \quad & -5y_1 + 8y_2 + 0.5y_3 \\ \text{subject to} \quad & 2y_1 + 4y_2 + y_3 \leq 10 \\ & -5y_1 + 6y_2 + y_3 \leq 1 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

This is converted this to Standard Form and thus into a tableau as follows,

ζ	y_1	y_2	y_3	w_1	w_2	Value
1	5	-8	-1/2	0	0	0
0	2	4	1	1	0	10
0	-5	6	1	0	1	1

For brevity and owing to the fact that we have solved Linear Programs of this form previously, we can skip the row operations and jump straight to the following optimal tableau,

ζ	y_1	y_2	y_3	w_1	w_2	Value
1	0	0	15/16	5/16	9/8	17/4
0	1	0	1/16	3/16	-1/8	7/4
0	0	1	7/32	5/32	1/16	13/8

Since we are minimising, although our ζ value will stay consistent, the y_i values will not correspond to our x_i which we wish to find. Instead, what we do is take the coefficients of the weight variables in our objective row and these correspond to x_i values. Hence, our solution set will be

$$x_1 = \frac{5}{16}, \ x_2 = \frac{9}{8}, \ \zeta = \frac{17}{4}.$$

3.4.1 Duality Theory

Following on from the example, it is now worth expanding upon how we approach these problems from a generalised viewpoint. The ideas and process of the Dual Simplex method stems from an area of Mathematical Optimization known as Duality Theory. The concepts within this field is worthy of a paper itself! So for brevity, we can just state key definitions and concepts that will come useful in understanding what we have done and for providing a sturdy theoretical footing going forward. This can first be done by considering how we can define a maximisation and a minimisation problem in tandem.

Definition 3.4. *Given a standard maximisation Linear Program:*

$$\begin{aligned} \textit{Maximise}\ & \mathbf{c}^T\mathbf{x} \\ \textit{Subject to:}\ & A\mathbf{x} \leq \mathbf{b} \\ & x_{1,..,n} \geq 0 \end{aligned}$$

We can express a corresponding Dual Linear Program as,

$$\begin{aligned} \textit{Minimise}\ & \mathbf{y}^T\mathbf{b} \\ \textit{Subject to:}\ & \mathbf{y}^T A \geq \mathbf{c}^T \\ & y_{1,..,m} \geq 0. \end{aligned}$$

Notice how transposing our examples in the previous section links these!

We call our initial Linear Program the Primal problem and its counterpart the Dual problem (In this case maximisation is the Primal and minimisation the Dual). We can also take the Dual of the Dual problem and this will get back to the primal. Essentially stating that we can always convert one form to the other. Another property connecting the Dual and Primal is the following:

$$\begin{aligned} \text{Minimise}\ & \mathbf{y}^T\mathbf{b} \\ \text{Subject to:}\ & \mathbf{y}^T A \geq \mathbf{c}^T \\ & y_{1,..,m} \geq 0 \end{aligned} \quad = \quad \begin{aligned} -\text{Maximise}\ & (-\mathbf{y})^T\mathbf{b} \\ \text{Subject to:}\ & (-\mathbf{y})^T A \leq -\mathbf{c}^T \\ & y_{1,..,m} \geq 0 \end{aligned}$$

We see here that the Primal and Dual are negations of each other in this situation. For some scenarios, this provides a useful tool in making it easier to compute the Simplex Method compared to changing the variable basis. Other, more formal properties connecting themselves to each other also exist; none so being more important than the Weak Duality Theorem defined below.

Theorem 3.1 (Weak Duality Theorem)**.** *If* $\mathbf{x}$ *is feasible for the standard maximisation problem and* $\mathbf{y}$ *is feasible for the Dual Problem, then*

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T\mathbf{b}.$$

Proof. Using the definitions from (3.4) it follows that

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T A\mathbf{x} \leq \mathbf{y}^T\mathbf{b}$$

□

What we are saying here is that the solution to the Dual problem gives us an upper bound on what our initial Primal problem solution could be. Furthermore, there is a corresponding Strong Duality concept complementing this theorem

Corollary 3.1.1. *If there exists feasible solutions for the Primal (x^*) Maximisation and its Dual (y^*) Minimisation problem, such that $\mathbf{c}^T\mathbf{x}^* = \mathbf{y}^{*T}\mathbf{b}$. Then both are optimal for their respective problems.*

Proof. Define $\mathbf{x}$ as a feasible solution for a maximisation problem, then from Theorem 3.1 we have,

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^{*T}\mathbf{b} = \mathbf{c}^T\mathbf{x}^*.$$

This shows that our x^* is optimal. [8]
Similarly, for y^* being optimal we have

$$\mathbf{y}^T\mathbf{b} \geq \mathbf{c}^T\mathbf{x}^* = \mathbf{y}^{*T}\mathbf{b}.$$

□

The final excursion in the Duality Theory field can follow on from this Strong Duality corollary, that is complementary slackness. Here we are showing a way to connect and dictate how our weight variables from Linear Programs we encounter can help us understand the optimality of said problem.

Theorem 3.2 (Complementary Slackness Theorem)**.** *Suppose x and y be feasible solutions for the primal and its dual problem. Also denote w and z as the Primal and Dual's slack variables. Then x^* and y^* are optimal for their problems IFF*

$$\mathbf{xz} = 0,$$
$$\mathbf{yw} = 0.$$

Proof. Recall from Theorem 3.1, we have that

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T A\mathbf{x}. \tag{3.4}$$

Here, if we write this out in summation form, we attain

$$\sum_j c_j x_j \leq \sum_j \left(\sum_i y_i a_{ij}\right) x_j.$$

From this we see that each term in the LHS is dominated by its corresponding RHS term, this is of consequence to the following:

$$c_j \leq \sum_i y_i a_{ij}.$$

Hence, we can say that (3.4) will be an equality IFF for every $j = 1, ..., n$, either $x_j = 0$ or the above holds but since

$$z_j = \sum_i y_i a_{ij} - c_{ij}$$

we see that the alternative to $x_j = 0$ is simply $z_j = 0$. For the second statement we can again refer to the following inequality from Theorem 3.1, i.e.

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T\mathbf{b}$$

and then solving using a same argument as earlier. [6] □

All in all, we have shown with these properties, the connection between maximisation and minimisation problems. Given this, we can see how in future, manipulating potentially very complex Linear Programs can be easier than on first inspection utilising a Linear Programs Dual as well as Duality Theory in general.

4 Interior Points Method

In this section, we shall explore an alternative way in solving Linear Problems. Primarily it is worth asking, why does one need another way of optimising? What problems are currently faced by the Simplex Method? We know when calculating iterations of the Simplex Method, we pivot between vertices till we find an optimal solution. But what if we encounter such a shape that has many vertices? We could be spending a very long time trying to carry out iterations in finding the optimal node. Interior points, however, work by wandering through the feasible region cutting through the middle till it reaches this vertex. We can start by introducing a popular interior point method and its more intuitive offspring.

4.1 Karmarkar's Algorithm

There is a plethora of interior points methods which can be utilised to solve Linear Programs (and nonlinear for that matter) yet for simplicity and ease of comparison, it is in the best interests of this paper to explore the first big method in this classification to be used, Karmarkar's algorithm. The method itself was developed by the aforementioned Narendra Karmarkar in 1984 as an alternative to the monopoly Dantzig's Simplex Method had in the solving of Linear Programs. Citing issues with how the Simplex Method could take up to exponential time[4](reasons why will be covered later in this section) in solving certain problems, Karmacker developed an algorithm which could take only polynomial time and be much more efficiant for all classifications of Linear Programs [9]. This algorithm itself (which comes from the paper *A New Polynomial-Time Algorithm For Linear Programming* [10]) is rather long and tedious. Luckily, a simpler version of the algorithm, Affine Scaling, was developed a year later. For small and simple problems, which we shall consider in this chapter, the method works just as efficiently as the original algorithm it stems from.

4.1.1 Affine Scaling method

Suppose we have a Linear Program of the following form:

$$\begin{aligned} \text{Minimise } & \mathbf{c^T x} \\ \text{Subject to: } & A\mathbf{x} = b \\ & x_{1,..,n} \geq 0 \end{aligned} \tag{4.1}$$

Alongside the following assumptions:

- There exists an optimal solution
- There exists a possible feasible solution
- The rows in A are linearly independent
- $\mathbf{c}$ is not a linear combination of rows in A.

[4]This is to say the method had time complexity of $O(2^n)$

Parameters, $\beta \in (0,1)$ and $\epsilon > 0$ are introduced, alongside an initial vector $\mathbf{e} = [1,....,1]^T$. An initial feasible solution $\mathbf{x}_0$ is also required. It should be noted that explanations for these variables can be found in Appendix A. with these parameters, we now have the following algorithm:

Algorithm 1: Primal Affine Scaling

Input: A, b, c, x_0, e, β, ϵ

k :=0

while *not optimal* **do**

- $X_k := \text{diag}(x_{k_1}, ..., x_{k_n})$
- $p_k := (AX_k^2A^T)^{-1}AX_k^2c$
- $r_k := c - A^T p_k$
- **if** $(-X_k r_k \geq 0)$ **then**
  - **return** *unbounded*
- $\gamma := \mathrm{e}^T X_k r_k$
- **if** $(r_k \geq 0 \;\&\&\; \gamma < \epsilon)$ **then**
  - optimal := **True**
- **else**
  - $x_{k+1} := x_k - \beta \frac{X_k^2 r_k}{||X_k r_k||_\infty}$
  - k :+= 1

[11][12]

What we can see here is our algorithm is starting at a predefined initial point, $\mathbf{x_0}$, and calculating which direction to move within our space. We check whether this could lead to an unbounded problem by calculating the null space and seeing if this is a positive quantity. Whilst our stopping condition needs to be set to a very small number, so we know that the direction and quantity we are moving in is arbitrarily small to care about. If the stopping condition was not present, then we could potentially be infinitely iterating towards the optimum. To show this algorithm in practice, since in this section we are trying to highlight practical alternatives to the Simplex Method, it will be worth looking at an example.

4.1.2 Example

Consider the following:

$$\begin{aligned} \text{Minimise} \quad & 2x_1 + 3x_2 \\ \text{Subject to:} \quad & x_1 + 2x_2 + 3x_3 = 6 \\ & x_2 + 4x_3 = 5 \\ & x_{1,2,3} \geq 0 \end{aligned}$$

If we compare this to our standard form we are dealing in (4.1), we can identify our A, b and c as

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \end{bmatrix}, \ \mathbf{b} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}, \ \mathbf{c} = \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix}.$$

It is evident that the rows in A are linearly independent as well as not having a linear combination to attain $\mathbf{c}$. Next, by taking our $\mathbf{e} = [1\ 1\ 1]^T$ and an initial feasible solution to be $\mathbf{x_0} = [1\ 1\ 1]^T$ (we can check this is true by inputting this into our Linear Program) with $\zeta_0 = 5$; then most inputs required for the algorithm have now been defined; except the step length which we can choose to define later on as well as our stopping condition ϵ, which can arbitrarily be defined as $\epsilon = 0.00001$ for the purpose of this exercise.

Taking $k = 0$ and carrying out the first steps of defining our diagonal matrix and calculating p_0 yields:

$$\begin{aligned}
X_0 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
p_0 &= (AX_0^2A^T)^{-1}AX_k^2\mathbf{c} \\
&= \left(\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 4 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \frac{17}{42} & -\frac{1}{3} \\ -\frac{1}{2} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 8 \\ 3 \end{bmatrix} \\
&= \begin{bmatrix} \frac{47}{21} \\ -\frac{5}{3} \end{bmatrix}
\end{aligned}$$

With this, we can then calculate r_0,

$$\begin{aligned}
r_0 &= \mathbf{c} - A^T p_0 \\
&= \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} \frac{47}{21} \\ -\frac{5}{3} \end{bmatrix} \\
&= \begin{bmatrix} -\frac{5}{21} \\ \frac{4}{21} \\ -\frac{1}{21} \end{bmatrix}.
\end{aligned}$$

Unboundedness and optimality checks can also be made at this stage:

$$\text{Unboundedness}: \; -X_0 r_0 = -\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}-\frac{5}{21}\\ \frac{4}{21}\\ -\frac{1}{21}\end{bmatrix} = \begin{bmatrix}\frac{5}{21}\\ -\frac{4}{21}\\ \frac{1}{21}\end{bmatrix} \not\geq 0$$

$$\text{Optimality}: \; r_0 \not\geq 0.$$

Finally, after passing both checks, we know we are still in our algorithm and we can now iterate by taking a step size of a fairly large $\beta = 0.95$[5] and calculate $\mathbf{x_1}$.

$$\begin{aligned}\mathbf{x_1} &= \mathbf{x_0} - \beta\frac{X_0^2 r_0}{||X_0 r_0||_\infty}\\ &= \begin{bmatrix}1\\1\\1\end{bmatrix} - \frac{0.95}{5/21}\begin{bmatrix}-\frac{5}{21}\\ \frac{4}{21}\\ -\frac{1}{21}\end{bmatrix}\\ &= \begin{bmatrix}1.95\\0.24\\1.19\end{bmatrix}\end{aligned}$$

By putting this into our Objective Function, $\zeta_1 = 4.62$, which you can see is an improvement towards a minimum as our objective value is decreasing. For brevity, if the reader was to carry out further iterations, we attain:

$$\mathbf{x_2} = \begin{bmatrix}2.235\\0.012\\1.247\end{bmatrix} \qquad \zeta_2 = 4.506 \qquad \beta = 0.95$$

$$\mathbf{x_3} = \begin{bmatrix}2.24925\\0.0006\\1.24985\end{bmatrix} \qquad \zeta_3 = 4.5003 \qquad \beta = 0.95$$

$$\ldots$$

$$\mathbf{x_7} = \begin{bmatrix}2.25\\0\\1.25\end{bmatrix} \qquad \zeta_7 = 4.5 \qquad \beta = 0.95$$

[5] We need to be careful with this number and other examples may require tweaks to this number. For this example, however, this number provides big steps without serious issues arising.

We reach the optimality condition at the 7^{th} iteration, yielding a vector of $\mathbf{x} = [2.25\ 0\ 1.25]$ and optimal value $\zeta = 4.5$[6]. We can also a visual look at how Affine Scaling traverses its way to a solution below. To avoid confusion, note that the axis x_1, x_2, x_3 are different from $X_{1,2,...,7}$ which denotes the progression of iterations. The former being our variables and the latter representing solution sets.

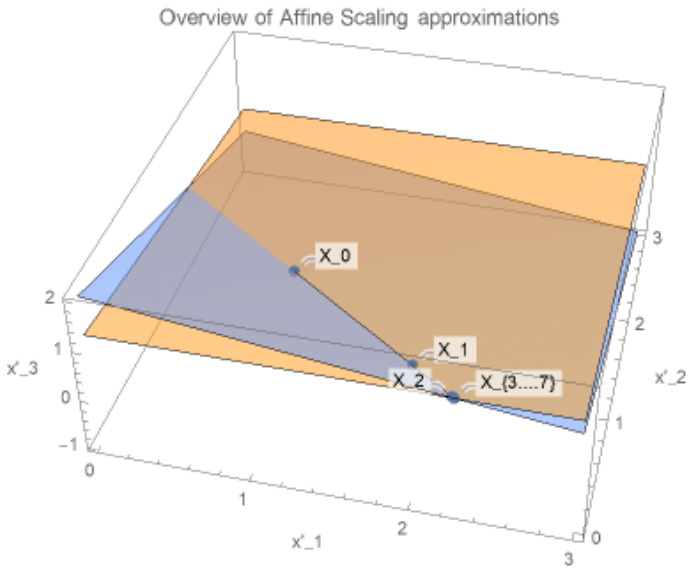


Figure 9: Affine Scaling overview of points.

[6]Code for calculating iterations can be found in Appendix B

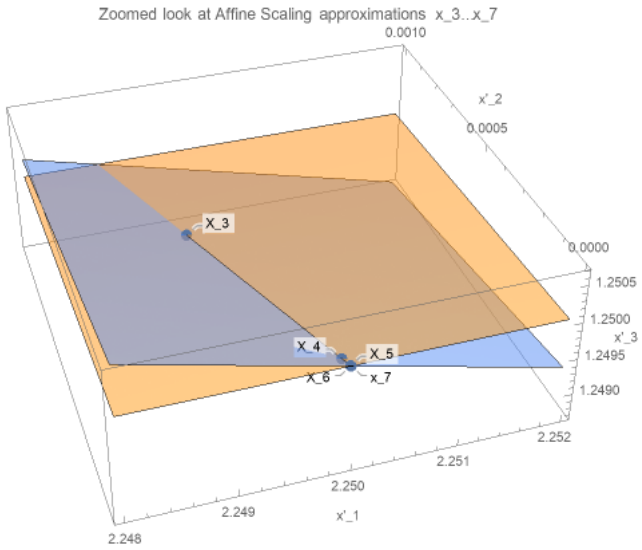


Figure 10: Affine scaling zoomed between (2.249,0.00004,1.24449) and (2.25,0,1.25)

As you can see with the points being so close together, we can make very large leaps towards an optimal solution with every iteration. Note that even when we zoom in we have a cluster of points, the steps taken are very small. From this, we can deduce why we bring in a stopping condition, that is because we would keep making these arbitrary small steps yet may never reach an exact value.

4.2 Interior Points Method Vs. Simplex Method

It would now be reasonable to question which method is better to use in today's world. However, it will be rather futile to simply suggest or assume one method reigns supreme in all areas than the other; instead, a better use of this section would be to lay down certain areas that these methods are important in and analyse which one is better for the specific situation. Two areas in which we can take an objective analysis is the practical implementation and speeds of respective methods. Although other qualifiers for comparisons exist, to keep with discussions already had in the paper, these are the key areas to investigate.

4.2.1 Programming Language implementation and Computational Speed

Our first two parameters can conveniently be done in the same analysis, this is by looking at "popular" implementations of Linear Programming and running these methods within different programming languages. As Python and Mathematica are the primary languages used in this paper we shall look at how each attempts to solve Linear Programs, alongside dedicated software packages used for optimisation.

Mathematica

Mathematica comes with a built in method, `LinearProgramming` in which it allows for one to "find a vector x that

minimizes the quantity $c \cdot x$ subject to the constraints m.$x \geq b$ and $x \geq 0$." [13] Multiple parameters are set, including one in which we are interested in, the method argument. This allows us to define if we want to calculate using Interior Points, Simplex, and Revised Simplex. Due to having not introduced the Revised Simplex Method in this paper, we can just focus on the first two. By default, Mathematica uses the Interior points method to solve Linear Programs. It is not clear however which specific Interior points method is implemented and with the fact this is proprietary software (or closed source) we cannot simply look at the source code to find out. Nevertheless, it should be worth looking into the speed at which we can solve Linear Programs using this built in function.

The `LinearProgramming` method allows us to calculate Linear Programs with large variables/constraints. For example, let's find the time it takes to calculate an arbitrary Linear Program with 1000 variables and 750 constraints. The code used and respective output can be seen below

```
In[1]= LinearProgramming[Range[1000],  SparseArray[{Band[{1, 1}] -> 1., Band[{1, 2}] -> 2.},
      {750,1000}],Range[750],Method->"InteriorPoint"];  \\ IPM
Out[1]= {0.021,Null}
In[2]= LinearProgramming[Range[1000], SparseArray[{Band[{1, 1}] -> 1., Band[{1, 2}] -> 2.},
     {750,1000}],Range[750],Method->"Simplex"]; \\ Simplex
Out[2]= {0.669,Null}
```

If we look at the respective outputs of 0.021 and 0.669, these indicate the time it takes to calculate each method and we can clearly see a time difference between the two. From this we can infer that for large computations in Mathematica, an interior points method approach will result in quicker computational timings. It should be noted that in the documentation, it states the interior points method only works for "machine-precision problems" [13]. This is an indication of a deficiency we have with such numerical approximation methods, so a tolerance, which we can see we used for Affine Scaling algorithm (Appendix A), is required. Unlike the Simplex Method which will give us an exact point of a vertices, the interior points method will essentially get us 'close enough' to a solution depending on the tolerance we give. For the purpose of Mathematica, this is machine precision (otherwise known as machine epsilon, ϵ_{mach}[7]).

Python

There is no native implementation of the methods in python. However, the SciPy library is a popular all encompassing suite for solving mathematical and engineering problems. In this library, we have a function `scipy.optimise.linprog` [14] which provides similar interface to the corresponding Mathematica optimisation function. Here again, the default method is the Interior points, with support for revised Simplex Method and the conventional Simplex Method (which is deemed legacy in the stable version cited, **v1.4.1**). We can see from this legacy statement that for Python, interior points is again preferred to the Simplex Method. However, in earlier implementations and versions, the Simplex Method was the original default implementation before expanding into others. Another note is, unlike the Mathematica code, SciPy is open source and thus it can be seen which interior method is used. Stated in the documentation, the Interior Points method is an implementation of the "MOSEK interior point optimizer"[15]. A unique solver, MOSEK uses a homogeneous model in contrast to a Primal-Dual based solver. Although the details of what this entails will not be covered here, in a brief sense one can think of this as a more computer friendly way to

[7] A common number in computer science defined as 2^{-52} for a double data type used by this method

solve Linear Programs using a numerical approach. With a significant focus on trying to consistently determine when a problem may be infeasible.

Although we have looked at only two examples in programming languages, it is clear that the preferred method for speed and usability is the Interior Points method. However, the availability of the Simplex Method in these functions highlights that the original authors of the respective code still find a use for its users to utilise this method.

4.2.2 Theoretical Comparisons

Although we have investigated the speed of respective algorithms in a practical sense, we can also study limitations and the theoretical speeds of such methods. As mentioned in **Section 4.1**, the Simplex Method can run at exponential time complexity ($O(2^n)$). This worst case scenario was conceived in a 1970 paper, which introduced the concept of a Klee-Minty Cube [16]. The problem, stated below in summation form, was pivotal in highlighting restrictions the Simplex Method could face with extreme Linear Programming cases.

$$\begin{aligned}
&\text{Maximise}: \sum_{j=1}^{n} 10^{n-j} x_j \\
&\text{Subject to}: 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq \sum_{j=1}^{i-1} 100^{i-1}, && (i = 1, ..., n) \\
&\qquad x_j \geq 0 && (j = 1, ..., n)
\end{aligned}$$

What this problem showed for the Simplex Method is that, for n variables, the iterations required to solve this is $2^n - 1$; indicating that every vertex has to be visited before the optimal one. For example, in the case $n = 3$, the Linear Program can be modelled as:

$$\begin{aligned}
&\text{Maximise}: 100x_1 + 10x_2 + x_3 \\
&\text{Subject to}: x_1 \leq 1 \\
&\qquad 20x_1 + x_2 \leq 100 \\
&\qquad 200x_1 + 20x_2 + x_3 \leq 10,000
\end{aligned}$$

This scenario takes 7, or $2^3 - 1$ iterations and proof can be found in Appendix C.

Looking at interior points, alongside the Karmakars algorithms polynomial time algorithm, the method that we have been most interested in throughout this section, Affine Scaling, does not have a proof to show polynomial time; in this case, the algorithm is designated super linear convergence[17]. However, it was shown that interior points algorithms *could* fall into the same trap of Simplex Method with visiting every vertex and thus causing exponential time complexity[18]. What is important to take away from this is that it is not as rigorously proved in comparison to the Klee-Minty cube for the Simplex Method. On a consistent basis, interior points methods outperform the Simplex Method.

5 Integer and Non-linear Programming

In this last section, we step away from the Simplex's usual surroundings of Linear Programming, to explore its applications in different areas of operational research. When modelling certain problems, we may not always be faced with simply creating linear models. Instead, we perhaps want to optimise quantities that require integer solutions or maybe our Objective Function cannot be made into a linear equation and necessitates a quadratic or even cubic function. In solving such possibilities, adaptations may be required to the conventional Simplex Method we have used so far; perhaps even the Simplex Method cannot be utilised at all. However, it should be noted that although methods to solve the following problems exist outside the scope of the Simplex Method (Convex optimisation, Interior Points Methods etc.) and will be touched upon where required, the main interest in this section is for finding solutions that involve the Simplex Method, whether this be in the adaptation of the method or using it in conjunction with other algorithms.

5.1 Integer Programming

Consider a beer company looking to take supply to a local show or a production line selling tins of soup. If we was to construct Linear Programs to optimise issues in these areas, one could not simply deal with variables resulting in non integer quantities; selling 13.5 tins of soup or taking 31.41 beers is nonsensical! A solution to this could simply just solve a Linear Program with their preferred method without the integer condition, then instinctively round the solutions to solve the integer Linear Program. Although we could use this as a quick fix and may work in some scenarios, overall it is likely this will lead to optimality and feasibility issues. Instead we'd prefer a method in which we could rigorously and efficiently solve these problems. Before this we can first redefine our original canonical definition (3.1) and formally state what we want to accomplish with integer Linear Programs. This simple tweak is added in the below definition:

Definition 5.1. *A Linear Program is suitable for Integer Linear Programming if it is of the canonical form:*

$$\begin{aligned} &\textit{Maximise} && \mathbf{c}^T\mathbf{x} \\ &\textit{Subject to:} && A\mathbf{x} \leq \mathbf{b} \\ & && x_{1,..,n} \geq 0, \\ & && x_{1,...,n} \in \mathbb{Z}^n \end{aligned}$$

and corresponding standard form:

$$\begin{aligned} &\textit{Maximise} && \zeta - c^T\mathbf{x} = 0 \\ &\textit{Subject to:} && A\mathbf{x} + \mathrm{I}\mathbf{w} = \mathbf{b} \\ & && x_{1,...,n}, w_{1,..,m} \geq 0 \\ & && x_{1,...,n} \in \mathbb{Z}^n \end{aligned}$$

With this in mind, it is now worth progressing onto the technique of solving these problems most relevant to us,

that is to say a technique which utilises the Simplex Method in generating solutions as well as being intuitive. For this, we shall investigate utilising the Branch and Bound method.

5.1.1 Branch and Bound method

To better understand how this works we shall run through a simple example, then generalise the process. So we can firstly consider the following:

Example 4. Solve the following Integer Linear Program:

$$\begin{aligned} \text{maximise} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & -x_1 + 4x_2 \leq 7 \\ & 2x_1 + 6x_2 \leq 15 \\ & x_1 + x_2 \leq 4 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Initially, we can solve this problem without the integer constraints. Giving us a conventional Linear Program we are comfortable in solving. In doing so, the following "optimal" solution to the non-integer Linear Program is attained

$$x_1 = 2.25, \quad x_2 = 1.75, \quad \zeta = 15.5.$$

This is known as our *relaxed* solution. From this, we can create our first node. Our initial node will contain the Upper Bound and Lower bound to a solution. The upper bound is assigned our relaxed solution whilst we round both these values down to attain a lower bound; updating the ζ value accordingly. Thus, we have the beginning of our diagram below:

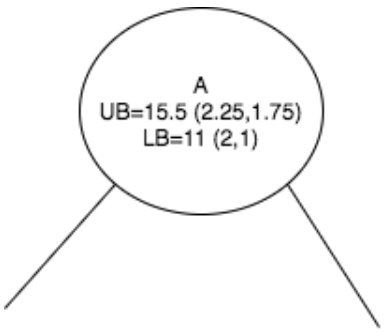


Figure 11: Branch and Bound at node A

Next, we choose the variable who's value is furthest away from the lower bound. In this case, x_1 is 0.25 away and x_2, 0.75 away. With x_2 being farther away, we can fix an additional parameter to our Linear Program using this variable.

What these parameters will be corresponds to a branch we go off from the initial node and these new values added will be rounded (up and down) from the relaxed solution. In briefer terms, these are $x_2 \leq 1$ and $x_2 \geq 2$. Note that in all cases, the lower value will use a "≤" inequality and the upper will use a "≥" inequality. We can then solve respective relaxed solutions again with the Simplex Method. i.e our new Linear Programs will be:

$$\begin{aligned} \text{maximise} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & -x_1 + 4x_2 \leq 7 \\ & 2x_1 + 6x_2 \leq 15 \\ & x_1 + x_2 \leq 4 \\ & x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned} \quad , \quad \begin{aligned} \text{maximise} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & -x_1 + 4x_2 \leq 7 \\ & 2x_1 + 6x_2 \leq 15 \\ & x_1 + x_2 \leq 4 \\ & x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

For a late reminder into setting this up to apply the Simplex Method, we will get the following standard forms set up to solve these relaxed solutions again:

$$\begin{aligned} \zeta - 3x_1 - 5x_2 &= 0 \\ -x_1 + 4x_2 + w_1 &= 7 \\ 2x_1 + 6x_2 + w_2 &= 15 \\ x_1 + x_2 + w_3 &= 4 \\ + x_2 + w_4 &= 1 \end{aligned} \quad , \quad \begin{aligned} \zeta - 3x_1 - 5x_2 &= 0 \\ -x_1 + 4x_2 + w_1 &= 7 \\ 2x_1 + 6x_2 + w_2 &= 15 \\ x_1 + x_2 + w_3 &= 4 \\ + x_2 + \alpha_1 - w_4 &= 2 \end{aligned}$$

Solving this results in new solution sets (for nodes B and C say), these correspond to:

$$\begin{aligned} &\underline{\text{Node B:}} \ (x_1, x_2) = (3, 1) \ \ \zeta = 14 \\ &\underline{\text{Node C:}} \ (x_1, x_2) = (1.5, 2) \ \ \zeta = 14.5 \end{aligned}$$

We can add these to our branch and bound diagram, alongside the lower bounds which are generated using the process we did for node A, to attain

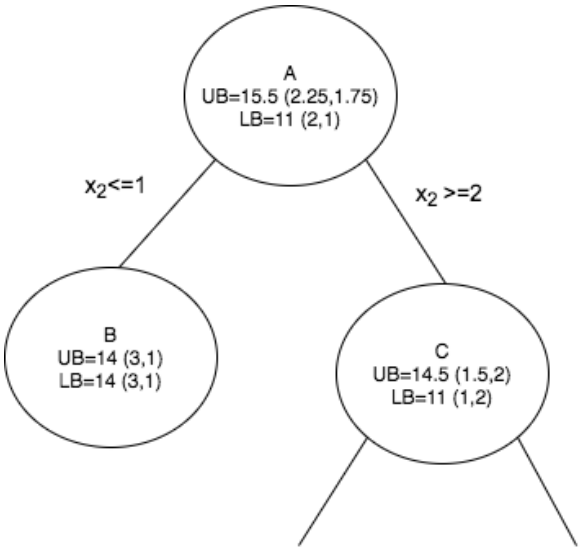


Figure 12: Branch and Bound with nodes B and C added

We can clearly see we have integer solutions at node B and so we have reached the end of this branch. Visually let us look at how our feasible region has changed and where our new solution for node B lies:

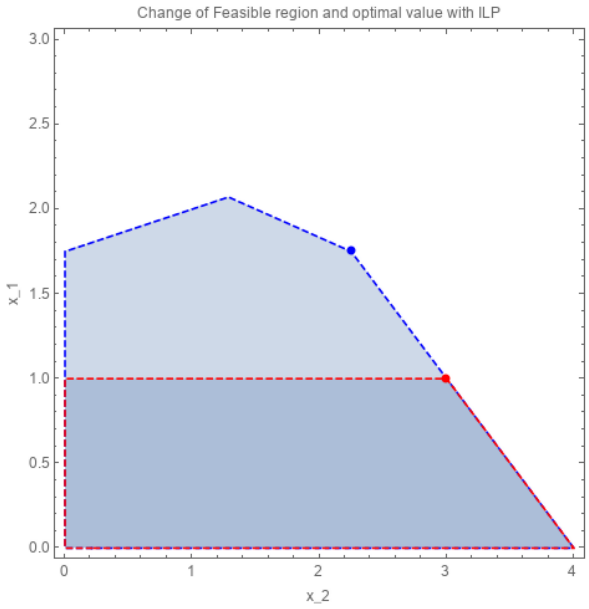


Figure 13: The original feasible region is in blue, with the optimum highlighted by the blue dot, the new ILP and its optimal is seen in the shrunken red area and the corresponding red dot.

One might now ponder if this means we have the optimal solution at node B? Well what we can do is look at branching off C and seeing what values we get. At node C, it should be clear x_1 will be where we add our new constraints since it's furthest away from the previous whole number. We can then add parameters $x_1 \leq 1$ and $x_1 \geq 2$ respectively to our Linear Program. Formally, we can solve these two relaxed programs:

$$\begin{aligned} \text{maximise} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & -x_1 + 4x_2 \leq 7 \\ & 2x_1 + 6x_2 \leq 15 \\ & x_1 + x_2 \leq 4 \\ & x_2 \geq 2 \\ & x_1 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned} \quad , \quad \begin{aligned} \text{maximise} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & -x_1 + 4x_2 \leq 7 \\ & 2x_1 + 6x_2 \leq 15 \\ & x_1 + x_2 \leq 4 \\ & x_2 \geq 2 \\ & x_1 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Notice from the RHS linear equation we cannot have that $x_1, x_2 \geq 2$. Doing so wouldn't satisfy the constraint $2x_1 + 6x_2 \leq 15$. For the LHS, if we investigate its feasible region we have a single point (1,2). This is a rather peculiar case which gives us a optimal value $\zeta = 13$. With this integer result, we see this is not a better option in comparison to Node B, coupled with the fact we have an integer solution, so we need not branch off from this node and we can now see our final diagram as:

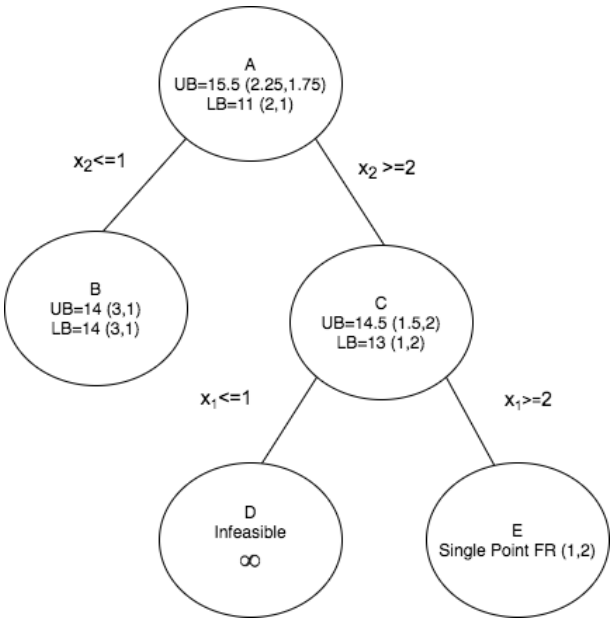


Figure 14: Branch and Bound with nodes D and E added

We can state our solution to the Integer Linear Program as the one attained at Node B, i.e.

$$\zeta = 14$$
$$(x_1, x_2) = (3, 1)$$

Although perhaps trivial to state, even though we have calculated some Linear Programs in this example by inspection and graphical understanding, it should not put the reader off the idea that this method doesn't rely on the Simplex Method. For computational calculation or even for a mathematical verification, the Simplex Method is integral in solving to attain exact solutions. Now, with this example, we can now formally declare the steps of the branch and bound method:

Branch and Bound Method for maximisation

1. Find the relaxed solution (i.e. without integer constraints).
2. Fix an initial node with the relaxed solution as an upper bound, and round down to attain a lower bound.
3. Chose the variable from the relaxed solution farthest away from the lower bound. Create subsequent boundaries for this variable by choosing the next and previous integer. (e.g. 3.14 will have 3 and 4 as "≤" and "≥" constraints respectively).
4. Create nodes for these new constraints and find the relaxed solutions with these constraints added to the Linear Program.
5. These new solution(s) are the new upper bounds, with the greatest integer solution (regardless where this node is) will be our lower bound.
6. If we reach a feasible integer solution with this upper bound, then our optimal integer solution has been found.
7. Otherwise, repeat the branching steps.

[19]

A final note on solving the corresponding minimisation problem. We round up (as opposed to rounding down) our relaxed solutions to essentially reverse the role of upper and lower bounds we've seen thus far.

5.1.2 Alternatives to Branch and Bound

The Branch and Bound method provides a useful technique into ensuring we can attain reliable integer solutions to Linear Programs, whilst harnessing the Simplex Method in the process. Outside this scope, other methods such as *heuristic methods* exist to provide an alternative way to solve such Integer Linear Programs. Whilst branch and bound fall under the category of an exact algorithm, heuristic methods are ways to solve problems by working towards an answer that can be deemed "good enough". In general terms, we can think of using these techniques to find solutions quicker at the detriment of sacrificing precision but ensuring we have a solution that works well

enough. All in all, some cases may not care if an absolute optimal is computed, but one in which a solution provides an apt alternative. We can apply this to integer Linear Programming in a sense that exact solution methods may take unrealistic computational power to produce results; so an heuristic method such as using Neural networks could be utilised (relevant paper on these topics for further reading can be found at [20]). The downside of these heuristic methods are the inability to find infeasible solutions.

5.2 Quadratic and higher dimensional Programming

Consider the following definition of a quadratic program,

Definition 5.2. *We can consider a problem to be a quadratic programming problem if it is of form:*

$$\begin{aligned} \textit{Minimise } & \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T Q\mathbf{x} \\ \textit{Subject to: } & A\mathbf{x} \geq \mathbf{b} \\ & x_{1,..,n} \geq 0, \end{aligned}$$

Note, the matrix Q, which should be new to us, is defined as a real symmetric matrix; the rest of the matrices/vectors should be familiar to us by this point. We see that quadratic programming still involves linear constraints, the key difference being our Objective Function is of a quadratic nature. To solve, a similar process, known as Wolfe's Method, can be utilised[21]. But why could we not use the Simplex Method to solve this? We can think about this intuitively, in the sense the Simplex Method solves by iterating through edges until it finds an optimal one. For the Objective Function being linear, we know that the maximum will be the only one. However with quadratic functions we have the issue of having both local and global optimal values which is something the Simplex won't be able to differentiate between. This is why adaptations or other methods must be used. This returns us to Wolfe's method. Since this is beyond our interests to cover in much detail, we can instead do a simple primer on what this involves and how it extends the Simplex Method.

Essentially, we have an algorithm in the similar sense to interior points due to us wanting to find a linear approximation. Like this method, we take an initial feasible approximation, x_0, which is to be derived using conventional Linear Programming techniques, i.e. Simplex. The Wolfe method would then attempt to make linear approximations by utilising partial derivatives of the Objective Function and working towards an optimal solution. Obvious caveats to this method comes with the problems faced by interior points methods, that this is an approximation so the solution may not be exact and just near enough. Hopefully, we can see how this extends the use of the Simplex Method to generate solutions. Uses of Quadratic programming in real world applications can be seen extensively in portfolio optimization[22]. This is particularly useful for when dealing with historic prices and the rate of change of portfolio assets. Here a quadratic function would need to be used rather than just expressing these in a linear sense.

When talking about problems involving cubic, quartic (and beyond), we can categorize this into the field of Polynomial Programming. Problems of higher degrees could be reduced down using substitutions to generate

quadratic programs. For example, in a cubic program, substitutions like ($x^{'} = x^2$) can convert the cubic inequality (x^3) to a quadratic of ($x^{'}x$). One should note that non-linear programs of this form is further out the scope of applicability and relevance to this paper, as compared to the already strained quadratic programming problem. With this area of optimization revolving around completely different ideas and fields of mathematics, such as algebraic geometry[23]. Alternatively, the Wolfe method could be extended to separate non-Linear Programs into linear approximations using partial derivatives and finding the rate of change. But, this would be time consuming and more efficient methods exist within other field of mathematics.

6 Conclusion and applications

From a simple introduction in terms of modelling real-life problems and applying the Simplex Method, to investigations into new methods and unique applications in different offshoots that branch from the conventional Linear Program; we have covered an interesting range of ideas to encapsulate the practicalities and importance of this area of study. Important definitions and theory have been touched upon to give a solid foundation in the subject. Allowing this to be built upon and applied to other areas of interest, such as Integer Programming and Duality Theory. We have also looked at problems graphically to understand visually what the Simplex Method and optimisation methods represent and how they traverse to optimality.

With computational analysis of the speed and efficiency, there was a peek into modern uses and applicability of the Simplex Method. Alongside the polynomial-time phenom of interior points algorithm and how the Klee-Minty cube caused havoc for the Simplex Method and gave a rather unflattering exponential time complexity. Analysis into how programming languages implemented optimisation methods highlighted the preference for interior points in the modern context, but the Simplex Method also provided a viable option and was available to utilise built-in functions for these languages.

Although a lot has been covered in this paper, there are also other areas in operational research and offshoots from the Simplex Method that, for whether it be due to brevity or the topic is beyond the scope of the target audience, have not been covered in this paper. One such topic of interest would be to comprehensively cover Duality Theory. The current excursion was brief and was not justifiable for the sheer amount of content and information that lie within the topic. One key example is Lagrangian duality, which is the analysis of the dual problem in non-linear forms. With another aspect being a more comprehensive look at how the theory builds into allowing us a straightforward method of switching between dual and primal problems.

Moving to the wider area of operations research, an interesting field in a modern context, is Convex optimization. It would be interesting to study this area which involves minimising convex shapes and functions. With widening applications in operations research and the bustling field of machine learning, it is a field worth pursuing.

A final area to touch on in this paper, given what we have learned in theory and real-world examples, is how can we use the Simplex Method in an advanced, real-world setting to achieve results, and can we do this dynamically? For one case, consider a supermarket and grocery store. In this setting, a key challenge of companies is to ensure shelves are sufficiently stocked. Whether this be for aesthetics or product availability, it is essential that stock is not wasted in the warehouse and can be made available for purchase at appropriate times. This is an optimization problem due to operational researchers and managers wanting a way to efficiently know when to stock shelves and track how much of each product is left. Given what we know so far about the Simplex Method, this should be a perfect fit for this problem and Linear Program. From the latter point of view, modelling the multitude of variables such as supply, demand, special offers, workers available and a host of other variables, we have seen in **Section 2** that modelling problems into Linear Programs can be a rather straightforward affair and simplify an endlessly complicated initial problem. We can optimise many different areas, such as cost, workflow and sales. For applying Simplex, we know the scale it can handle and the ease of implementation, problems such as these is where the Simplex Method still reigns supreme. Whilst in almost all cases, except the rare chance the Linear Program becomes something akin to

the Klee-Minty problem, solving this program will be a relatively efficient process. Further adaptations to create a more dynamic workflow could come in the form of nesting Simplex. For example a supermarket could split items into categories, optimise which types of items need prioritization, then further calculate how best to distribute items within these categories. Working out how much more time worthy this method could be, as compared to doing one large Linear Program with many variables and constraints would be worth investigating in the future.

References

[1] Donald J. Albers and Constance Reid. An interview with george b. dantzig: The father of linear programming. *The College Mathematics Journal*, 17(4):292–314, 1986.

[2] Arjang A. Assad and Saul I. Gass, editors. *Profiles in operations research: pioneers and innovators*. Number 146 in International series in operations research management science. Springer US, New York, 2011. OCLC: 756277845.

[3] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, Chichester, reprinted edition, 2000. OCLC: 247967491.

[4] Robert E. Bixby. A brief history of linear and mixed-integer programming computation. 2012.

[5] Kannan T.R., G. Dinakaran, and N.J. Lavanya. A graphical approach for solving three variable linear programming problems. 03 2004.

[6] Robert J. Vanderbei. *Linear programming: foundations and extensions*. Springer, New York, 2013.

[7] *Understanding and Using Linear Programming*. Universitext. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[8] Thomas S. Ferguson. *Linear Programming, A concise introduction*.

[9] Paris Kanellakis theory and practice award. `https://web.archive.org/web/20120402212312/http://awards.acm.org/citation.cfm?id=0424282&srt=all&aw=147&ao=KANELLAK&yr=2000`, April 2012.

[10] Narendra Karmarkar. A new polynomial-time algorithm for linear programming-ii. *Combinatorica*, 4:373–395, 12 1984.

[11] Shu-Cherng Fang and Sarat Puthenpura. *Linear Optimization and Extensions: Theory and Algorithms*. Prentice-Hall, Inc., USA, 1993.

[12] Robert J. Vanderbei, Marc S. Meketon, and Barry A. Freedman. A modification of karmarkar's linear programming algorithm. *Algorithmica*, 1(1-4):395–407, November 1986.

[13] Wolfram alpha linear programming documentation. `https://reference.wolfram.com/language/ref/LinearProgramming.html`.

[14] Scipy linear programming documentation. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html`.

[15] Erling Andersen and Knud Andersen. The mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm. 33, 01 1999.

[16] Victor Klee, George J Minty, and Oved Shisha. Inequalities, iii. *How Good is the Simplex Algorithm (1972)*, pages 159–175, 1972.

[17] Jeffrey C. Lagarias and Michael J. Todd, editors. *Mathematical Developments Arising from Linear Programming*, volume 114 of *Contemporary Mathematics*. American Mathematical Society, Providence, Rhode Island, 1990.

[18] Nimrod Megiddo and Michael Shub. Boundary behavior of interior point algorithms in linear programming. *Mathematics of Operations Research*, 14(1):97–146, 1989.

[19] Eva K. Lee and John E. Mitchell. *Integer programming: branch and bound methodsInteger Programming: Branch and Bound Methods*, pages 1634–1643. Springer US, Boston, MA, 2009.

[20] Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-GÃşmez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3):545–566, December 2016.

[21] Philip Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, 1959.

[22] Gerd Cottle, Richard W.and Infanger. *Harry Markowitz and the Early History of Quadratic Programming*, pages 179–211. Springer US, Boston, MA, 2010.

[23] Grigoriy Blekherman, Pablo A. Parrilo, and Rekha R. Thomas, editors. *Semidefinite optimization and convex algebraic geometry*. MOS-SIAM series on optimization. Society for Industrial and Applied Mathematics : Mathematical Programming Society, Philadelphia, 2013.

Appendices

A Variables for Affine Scaling algorithm

The (primal) Affine Scaling algorithm takes in a multitude of inputs. These, alongside explanations can be seen in the following table.

Variable	type	Explanation
A	Matrix	Coefficient matrix for LHS of linear constraints.
b	vector	The RHS of linear constraints.
c	vector	coefficient matrix for Objective Function.
$\mathbf{x_0}$	vector	An initial guess for our approximation that is a basic feasible solution for a Linear Program.
e	vector	constant vector of ones, the length is dependent on the number of variables within the Linear Program. I.e. if a Linear Program has variables x_1, x_2, x_3, then the vector **e** will be $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$
β	constant	Real number between 0 and 1 to decide the step size we take towards a solution. As often is the case with numerical approximations, issues may arise when taking certain extreme lengths[6].
ϵ	constant	Small constant to determine the leeway we have with accuracy towards a solution. A very small number may appear more precise and accurate, however it will take more computational power with more iterations compared to a larger (but still small) ϵ.

B Primal Affine Scaling algorithm code

The following is Python code to compute the optimal value using the primal affine scaling method on a Linear Program adhering to conventions stated in 4.1. Values are hard coded into the program so requires manual change to compute a different Linear Program

```
# Code to produce a simple computation of the primal affine scaling algorithm
import numpy as np
import numpy.linalg as npla

def iterate(A,b,c,x_0,beta):
  # Set some intial quantities as per algorithm
  k=1
  e=np.ones(len(A[0])).transpose()

  # Keep check for optimality
  optimal=False

  # Our first iteration
  x_k=x_0

  # The halting threshold
  eps=0.00001

  # Keep iterating till an optimality check passes
  while not optimal:
    # Diagonal matrix and its square
    x_diag = np.diag(x_k)
    x_diag_sq=np.square(x_diag)

    # Our P_k
    p_k=npla.multi_dot([npla.inv(npla.multi_dot([A,x_diag_sq, A.transpose()])),A,x_diag_sq,c
    ])

    # Our r_k
    r_k=c-A.transpose().dot(p_k)

    # Optimality check
    gamma=npla.multi_dot([e,x_diag,r_k])
    if(all(i >= 0 for i in r_k) and gamma <= eps):
      optimal=True
      # Print some useful information to the user
      print('Optimal Reached!!')
      print('x={}'.format(x_k))

      # Output a rounded result to avoid confusing standard form output
      print('zeta={}'.format(round(np.dot(c,x_k),2)))
    else:
      # Calculate next iteration
      x_k_next=x_diag_sq.dot(r_k)
```

```
      infnorm=npla.norm(x_diag.dot(r_k),np.inf)
      x_k=x_k-(beta/infnorm)*x_k_next

      # Print useful info about this iteration
      print('Iteration {}'.format(k))
      print('============')
      print('x_{}={}'.format(k,x_k))
      print('zeta={}'.format(np.dot(c,x_k)))
      print('Distance from solution: {}'.format(gamma))
      print('~~~~~~~~\n')
    k=k+1

# Program starts running in this state
if __name__ == '__main__':
  # Define our variables, change these if you want to try different inputs
  A=np.array([[1, 2, 3], [0,1,4]], dtype='f')
  b=np.array([6, 5])
  c=np.array([2, 3, 0])
  x_0=np.array([1,1,1])
  param=0.95

  # Carry out algorithm
  iterate(A,b,c,x_0,param)
```

Listing 1: Primal Affine Scaling method

C Klee-Minty cube for n=3

Consider the following Linear Program:

$$\begin{aligned} \text{Maximise}: \quad & 100x_1 + 10x_2 + x_3 \\ \text{Subject to}: \quad & x_1 \leq 1 \\ & 20x_1 + x_2 \leq 100 \\ & 200x_1 + 20x_2 + x_3 \leq 10,000 \end{aligned}$$

Visually, we can see this shape below,

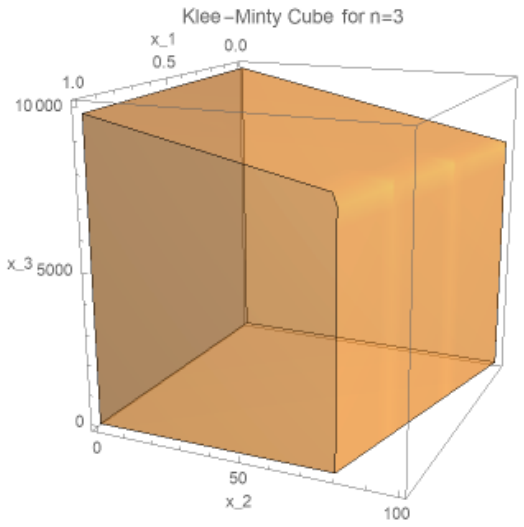


Figure 15: Klee-Minty Cube for n=3, we expect the Simplex Method to visit all nodes till it reaches an optimal value

From the Linear Program we can attain the standard form and thus a Simplex Tableau,

$$\begin{aligned} \zeta - 100x_1 - 10x_2 - x_3 &= 0 \\ x_1 + w_1 &= 1 \\ 20x_1 + x_2 + w_2 &= 100 \\ 200x_1 + 20x_2 + x_3 + w_3 &= 10,000 \end{aligned} \quad \Longrightarrow$$

ζ	x_1	x_2	x_3	w_1	w_2	w_3	Value
1	-100	-10	-1	0	0	0	0
0	1	0	0	1	0	0	1
0	20	1	0	0	1	0	100
0	200	20	1	0	0	1	10000

Carrying out iterations to an optimal solution can be seen below:

ζ	x_1	x_2	x_3	w_1	w_2	w_3	Value
1	-100	-10	-1	0	0	0	0
0	(1)	0	0	1	0	0	1
0	20	1	0	0	1	0	100
0	200	20	1	0	0	1	10,000
Iteration 1							
1	0	-10	-1	100	0	0	100
0	1	0	0	1	0	0	1
0	0	(1)	0	-20	1	0	80
0	0	20	1	-200	0	1	9800
Iteration 2							
1	0	0	-1	-100	10	0	900
0	1	0	0	(1)	0	0	1
0	0	1	0	-20	1	0	80
0	0	0	1	200	-20	1	8200
Iteration 3							
1	100	0	-1	0	10	0	1000
0	1	0	0	1	0	0	1
0	20	1	0	0	1	0	100
0	-200	0	(1)	0	-20	1	8000
Iteration 4							
1	-100	0	0	0	-10	1	9000
0	(1)	0	0	1	0	0	1
0	20	1	0	0	1	0	100
0	-200	0	1	0	-20	0	8000
Iteration 5							
1	0	0	0	100	-10	1	9100
0	1	0	0	1	0	0	1
0	0	1	0	-20	(1)	0	80
0	0	0	1	200	-20	1	8200
Iteration 6							
1	0	10	0	-100	0	1	9900
0	1	0	0	(1)	0	0	1
0	0	1	0	-20	1	0	80
0	0	20	1	-200	0	1	9800
Iteration 7							
1	100	10	0	0	0	1	10000
0	1	0	0	1	0	0	1
0	20	1	0	0	1	0	1000
0	200	20	1	0	0	1	10000

As we can see this takes 7 iterations, which we expected, and results in an optimal solution set of

$$(x_1, x_2, x_3, w_1, w_2, w_3) = (0, 0, 10000, 1, 1000, 10000), \quad \zeta = 10000.$$

If we take solution sets at each iteration, in particular the (x_1, x_2, x_3) values, i.e.

$$\begin{aligned} \text{Initial Tableau}: (0,0,0) \quad & \text{Iteration 1}: (1,0,0) \\ \text{Iteration 2}: (1,80,0) \quad & \text{Iteration 3}: (0,100,0) \\ \text{Iteration 4}: (0,100,8000) \quad & \text{Iteration 5}: (1,80,8200) \\ \text{Iteration 6}: (1,0,9800) \quad & \text{Iteration 7}: (0,0,10000) \end{aligned}$$

Looking back to Figure 15, we can see that these sets correspond to each vertex. Confirming our claim that the Simplex Method iterates through every vertex to get to the optimal solution under a Klee-Minty cube.