Week 1

Introduction and Basics

1.1 Introduction

This module concerns a particular branch of operational research involving *linear optimisation problems*. In these problems, we want to find the "best possible" plan, policy, allocation, etc. subject to a set of constraints that describe which plans, policies, etc. can actually be implemented. For historical reasons, the resulting optimisation problem is called a *linear program* and the process of formulating and solving the problem is called *linear programming*. Despite the terminology, this process does *not* involve what you and I would consider "computer programming." This module will *not* involve writing computer code or learning a programming language. Instead, the term is more similar to what we mean we say something like a "programme for production," in which we mean a *plan*, *scheme*, or *schedule*.

1.1.1 Linear Programming

In order to make this discussion more concrete, let's consider an example of a problem arising in the real world that we might want to solve:

Example 1.1. A university student is planning her daily food budget. Based on the British Nutrition Foundation's guidelines for an average female of her age she should consume the following daily amounts of vitamins:

Vitamin	mg/day
Thiamin	0.8
Riboflavin	1.1
Niacin	13
Vitamin C	35

After doing some research, she finds the following cost, calories, and vitamins (in mg) per serving of several basic foods:

Food	Cost	Thiamin	Riboflavin	Niacin	Vitamin C
Bread	£0.25	0.1	0.1	1.3	0.0
Beans	£0.60	0.2	0.1	1.1	0.0
Cheese	£0.85	0.0	0.5	0.1	0.0
Eggs	£1.00	0.2	1.2	0.2	0.0
Oranges	£0.80	0.2	0.1	0.5	95.8
Potatoes	£0.50	0.2	0.1	4.2	28.7

How can the student meet her daily requirements as cheaply as possible?

In the interest of simplicity, our example does not consider the requirements for several Vitamins like B6, B12, Folate, A, and D, as well as limits on calories, protein, fats, and sugars. However, this approach could easily be extended to the full case; one early application of linear programming did exactly this.

To model this problem, we introduce six variables x_1, x_2, x_3, x_4, x_5, and x_6 to express, respectively, the number of servings of bread, beans, cheese, eggs, oranges, and potatoes the student purchases daily. Then, the total daily cost can be written as:

$$0.25x_1 + 0.60x_2 + 0.85x_3 + 1.00x_4 + 0.80x_5 + 0.50x_6\,. \tag{1.1}$$

Now, we need to make sure that we eat enough of each of the 4 vitamins in our example. If we look at the nutritional table, the number of milligrams of Thiamin in the student's daily diet can be expressed as:

$$0.1x_1 + 0.2x_2 + 0.0x_3 + 0.2x_4 + 0.2x_5 + 0.2x_6$$

We want to make sure that this is at least 0.8. In other words, we want to choose x_1, x_2, x_3, x_4, x_5, and x_6 so that:

$$0.1x_1 + 0.2x_2 + 0.0x_3 + 0.2x_4 + 0.2x_5 + 0.2x_6 \geq 0.8\,. \tag{1.2}$$

We can do the same thing for each of the other three vitamins, to get the following 3 inequalities:

$$0.1x_1 + 0.1x_2 + 0.5_3 + 1.2x_4 + 0.1x_5 + 0.1x_6 \geq 1.1 \tag{1.3}$$
$$1.3x_1 + 1.1x_2 + 0.1x_3 + 0.2x_4 + 0.5x_5 + 4.2x_6 \geq 13 \tag{1.4}$$
$$0.0x_1 + 0.0x_2 + 0.0x_3 + 0.0x_4 + 95.8x_5 + 28.7x_6 \geq 35\,. \tag{1.5}$$

So, we want to select some non-negative values x_1, x_2, x_3, x_4, x_5, and $x_6$so that (1.1) is *minimised*, under the constraint that our values must satisfy inequalities (1.2), (1.3), (1.4), and (1.5). We write this formally as follows:

$$\begin{aligned} \text{minimize} \quad & 0.25x_1 + 0.60x_2 + 0.85x_3 + 1.00x_4 + 0.80x_5 + 0.50x_6 \\ \text{subject to} \quad & 0.1x_1 + 0.2x_2 + 0.0x_3 + 0.2x_4 + 0.2x_5 + 0.2x_6 \geq 0.8 \\ & 0.1x_1 + 0.1x_2 + 0.5x_3 + 1.2x_4 + 0.1x_5 + 0.1x_6 \geq 1.1 \\ & 1.3x_1 + 1.1x_2 + 0.1x_3 + 0.2x_4 + 0.5x_5 + 4.2x_6 \geq 13 \\ & 0.0x_1 + 0.0x_2 + 0.0x_3 + 0.0x_4 + 95.8x_5 + 28.7x_6 \geq 35 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned} \tag{1.6}$$

This is an example of what we call a *mathematical program*. Let's look at the parts in more detail:

Definition 1.1 (Mathematical Program)**.** A *mathematical program* represents an optimisation problem. It consists of 4 components:

1. A set of *decision variables* which may have *sign restrictions*. We give these as a list at the bottom of the program, and say which variables are non-negative (≥ 0), which variables are non-positive (≤ 0), and which variables can be either positive or negative (we say these last are *unrestricted*).
2. An *objective function*, that represents some function of the decision variables that we are trying to optimise. This appears in the first line of the program, together with the next component...
3. A *goal*, which specifies whether we are trying to maximise or minimise the objective function—that is, do we want to set the variables to make this function as large as possible or do we want to set them to make it as small as possible?
4. A set of *constraints*, given by equations and inequalities relating the decision variables. We are only allowed to choose values for our variables that make all of these inequalities and equations true. These appear after the phrase "subject to.".

Of course, the set of restrictions are also inequalities, so we could consider them as part of the *constraints*, but it will be convenient to treat these separately later on. When we say that a variable is *unrestricted*, we just mean that we don't care about whether it takes positive or negative values; it does *not* mean that it is allowed to take any arbitrary value, since the constraints of the program will probably rule some of these out.

NOTE: You should always list all variables at the end of your linear programs, and explicitly say whether they are ≥ 0, ≤ 0, or unrestricted.

You should think of a mathematical program as an optimisation problem that we want to solve. The goal of this problem is to assign values to the *decision variables* in order to make the *objective function* as large or as small as possible, depending on the stated *goal*. In calculus, you learned how to find maxima or minima of a given function by considering its derivative. Here, however, we have an extra complication, which is that we are only allowed to choose values for the decision variables that together satisfy all of the given *constraints* and *restrictions*.

We can represent the decision variables $x_1, \ldots, x_n$ as a vector $\mathbf{x}$. Then, when considering a mathematical program, we call a particular choice $\mathbf{x}$ of values a *feasible solution* to this mathematical program if and only if these values satisfy all of the variables' sign restriction and all of the constraints of the program (we will make this completely formal later). We can restate any mathematical optimisation problem written as above in plain English as "find a feasible solution $\mathbf{x}$ that optimises (i.e. maximises or minimises, depending on our goal) the objective function". We call a feasible solution that optimises the objective an *optimal solution*. Note that a problem may have no optimal solutions, a single optimal solution, or even more than one optimal solution.

The definition of a general mathematical program that we have given is very general: we could use any function as our objective and inequalities involving arbitrary functions of the variables as constraints. In this module, we consider only mathematical optimisation programs and problems that at are *linear*. We call these *linear programs* (they will be defined formally a bit later). We will see that the example problem we have considered is in fact a *linear program*. This means that the objective and constraints have a special structure that allows us to solve them efficiently: in this case, we can show that the best daily diet costs about £2.27, and consists of approximately 2.03 servings of bread, 0.54 servings of eggs, and 2.44 potatoes per day. This can be done by using a simple algorithm called the Simplex Algorithm.

1.1.2 Duality and Game Theory

After studying the Simplex Algorithm, we will move on to the notion of duality which will ultimately lead us to theories of 2-player games from economics. Let's

consider another example:

Example 1.2. Suppose you run a company that sells dietary supplements. You make four kinds of tablets for, respectively, Thiamin, Riboflavin, Niacin, and Vitamin C. You decide to market them to poor university students as a cheaper way to meet their daily vitamin requirements. How should you price your supplements to maximise the revenue you can obtain for one day's worth of supplements?

We can use a set of 4 variables y_1, y_2, y_3, and y_4 to represent the price you charge (in £) for 1mg of Thiamin, Riboflavin, Niacin, and Vitamin C, respectively. We know from the previous example, that each day a student will need to consume 0.8 mg of Thiamin, 1.1 mg of Riboflavin, 13mg of Niacin, and 35 mg of Vitamin C. So, we can write our overall revenue for one day's worth of tablets as:

$$0.8y_1 + 1.1y_2 + 13y_3 + 35y_4 \tag{1.7}$$

However, we need to be careful: we want to make sure that our supplements are no more costly than a regular diet, or else no one will buy them! We know from the previous example that a single serving of bread contains 0.1mg of Thiamin and Riboflavin, 1.3mg of Niacin and 0mg of Vitamin C, but costs only £0.25. So, we should make sure that this mixture of supplements is no more expensive than £0.25. In other words we want to set our prices so that:

$$0.1y_1 + 0.1y_2 + 1.3y_3 + 0.0y_4 \leq 0.25\,. \tag{1.8}$$

Similarly, we obtain an inequality for each of the remaining 4 foods in Example 1.1. Proceeding as before, we can write the problem as the following linear program:

$$\begin{aligned}
\text{maximize} \quad & 0.8y_1 + 1.1y_2 + 13y_3 + 35y_4 \\
\text{subject to} \quad & 0.1y_1 + 0.1y_2 + 1.3y_3 + 0.0y_4 \leq 0.25 \\
& 0.2y_1 + 0.1y_2 + 1.1y_3 + 0.0y_4 \leq 0.60 \\
& 0.0y_1 + 0.5y_2 + 0.1y_3 + 0.0y_4 \leq 0.85 \\
& 0.2y_1 + 1.2y_2 + 0.2y_3 + 0.0y_4 \leq 1.00 \\
& 0.2y_1 + 0.1y_2 + 0.5y_3 + 95.8y_4 \leq 0.80 \\
& 0.2y_1 + 0.1y_2 + 4.2y_3 + 28.7y_4 \leq 0.50 \\
& y_1, y_2, y_3, y_4 \geq 0
\end{aligned} \tag{1.9}$$

If we solve this LP, we find that the best set of prices give a total value of £2.27. This is exactly the same answer as our previous example! This is not a coincidence,

but a rather a consequence of *duality*. We will see that every minimisation LP has an associated maximisation LP called its *dual* and that their optimal solutions always have the same value. It turns out that (1.9) is the dual of problem (1.6), and so they must have the same optimal value.

We can think of this whole situation as a competition between our student and our vitamin salesman. Then the result is perhaps not so surprising: the daily cost of vitamins cannot be larger than the price of the diet we already found, or our student would not buy anything! Similarly, if the cost was much smaller than the diet, then we might expect to be able to increase the cost of some vitamin to make more money.

Building on these ideas, we will turn to the study of 2-player *games*. Here, we have 2 different agents and each can take some action from a given set. Both agents are trying to maximise their profit, but, in general the profit depends on the actions taken by *both agents*. Here is a concrete example of the sort of game we will study:

Example 1.3. At the end of the "Golden Balls" game show, two players compete for a prize pool of money. Each player has 2 options: they can choose to *split* the prize or *steal* the prize. Both select their choices in secret and then reveal them. If both players chose to *split*, then the prize is split equally between them. If both players chose to *steal*, then neither gets any money. However, if one player chose to *split* and the other to *steal*, then the player that chose *steal* gets all of the money (and the player that chose to *split* gets none).

You may recognise this as a variant of the classic "Prisoner's Dilemma." What do you think the optimal strategy is? It's not so clear for this game! In the last part of the module we will develop mathematical tools for dealing with such games and see precisely what makes this game so tricky. In general the study of these decision-making problems constitutes the basis for *game theory*. The study of game theory has led to 11 Nobel prizes in Economics and can be used to guide planning and decision-making in competitive settings.

1.2 Background on Linear Algebra and Notation

Before beginning our formal discussion of linear programming, let us review some basic notions from linear algebra and establish some conventions for notation when talking about matrices and vectors. You should be familiar with the material in this section, although some of the conventions may be slightly different than you are used to.

1.2.1 Vectors, Matrices, and Linear Combinations

We will denote vectors by using lower-case letters in **bold** and matrices by CAPITAL letters. All of our vectors and matrices will contain real numbers as their elements. We adopt the convention that all vectors are *column vectors* belonging to the Euclidean space $\mathbb{R}^n$ for some specified dimension n (usually n will be clear from our context). For example, $\mathbf{x}$ denotes a vector, which is given by:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \, . \end{pmatrix} \tag{1.10}$$

Note that the 1st entry of $\mathbf{x}$ is denoted x_1, the second x_2, and so on. So, as another example, if $\mathbf{y} = \begin{pmatrix} 2 \\ 4 \\ 1 \end{pmatrix}$ then $y_1 = 2$, $y_2 = 4$, and $y_3 = 1$. We will use $\mathbf{0}$ to denote the *all-zero* vector, which is simply a vector whose entries are all zero. The exact dimension (i.e. number of entries) of $\mathbf{0}$ will depend on the context in which it appears, but should always be clear.[1]

We denote the *transpose* of a matrix A by A^T. If A is an $n \times m$ matrix then A^T is an $m \times n$ matrix obtained by exchanging the rows and columns of A. For example, if

$$A = \begin{pmatrix} 2 & 3 & 1 \\ 1 & 4 & 5 \end{pmatrix} \text{ then } A^\mathsf{T} = \begin{pmatrix} 2 & 1 \\ 3 & 4 \\ 1 & 5 \end{pmatrix}$$

Note that we regard n-dimensional vectors as $n \times 1$ matrices. Then, transposing a (column) vector gives us a corresponding row column vector. For example if $\mathbf{x}$ is given by (1.10), then $\mathbf{x}^\mathsf{T}$ is a $1 \times n$ matrix given by:

$$\mathbf{x}^\mathsf{T} = (x_1, x_2, \ldots, x_n) \ .$$

Using this notation, we can denote the standard dot-product[2] of two n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$\mathbf{x}^\mathsf{T}\mathbf{y} = (x_1, x_2, \ldots, x_n) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

[1] Note that a similar problem happens with the identity matrix, which we usually just denote as I, even though there is a different $n \times n$ identity matrix for each value of n.

[2] In other contexts, this is also sometimes called a "scalar product" or "inner product."

In other books or modules, you may have seen this operation denoted $\mathbf{x}\cdot\mathbf{y}$ or $\langle \mathbf{x}, \mathbf{y}\rangle$. Here, we use $\mathbf{x}^\mathsf{T}\mathbf{y}$ since this agrees exactly with the standard rule for multiplying 2 matrices: specifically we can think of this operation as multiplying a $1 \times n$ matrix $\mathbf{x}^\mathsf{T}$ by a $n \times 1$ matrix $\mathbf{y}$, as above, to obtain a 1×1 matrix, which we regard as a value in $\mathbb{R}$. We will use $\cdot$ to denote standard scalar multiplication (that is, multiplication of 2 *numbers*).

This leads us to the following general notion:

Definition 1.2 (Linear Combination)**.** A *linear combination* of $x_1, x_2, \ldots, x_n$ is an expression of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n\,,$$

where $a_1, a_2, \ldots, a_n$ are each real numbers.

Note that we did not specified what types of mathematical objects $x_1, \ldots, x_n$ are in Definition 1.2. When $x_1, \ldots, x_n$ are all *numbers*, we can write linear combinations succinctly as $\mathbf{a}^\mathsf{T}\mathbf{x}$, where:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n\,. \end{pmatrix}$$

However, the definition applies also in the case where each x_i in $\{x_1, x_2, \ldots, x_n\}$ is a *vector* in $\mathbb{R}^m$. Switching to our standard (bold) notation for vectors, we have that a linear combination of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ is given by:

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots a_n\mathbf{x}_n,$$

where, again, each of the a_i in $\{a_1, a_2, \ldots, a_n\}$ is a real number. Note that the linear combination $a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n$ is now a new vector in $\mathbb{R}^m$. In this case, we can again represent the linear combination succinctly, as follows: as above, let $\mathbf{a}$ be an $n \times 1$ vector given by $\mathbf{a}^\mathsf{T} = (a_1, a_2, \ldots, a_n)$ but now let X be an $m \times n$ matrix whose jth column is given by $\mathbf{x}_j$. Then, the linear combination $a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n$ can be written as $X\mathbf{a}$.

1.2.2 Linear Equations and Inequalities

Definition 1.3 (Linear Equations and Inequalities)**.** We call an expression of the form:

$$a_1x_1 + a_2x_2 + \cdots a_nx_n = b,$$

where each $a_i \in \mathbb{R}$ and $b \in \mathbb{R}$ are constants a *linear equality* over $x_1, x_2, \ldots, x_n$. If instead of $=$ we have $\leq$, or $\geq$ above, we call the expression a *linear inequality*.

Note that a linear equation or inequality simply relates a *linear combination* or variables $x_1, x_2, \ldots, x_n$ to some constant b. If we write the variables x_i and constants a_i in Definition 1.3 as vectors $\mathbf{x}$ and $\mathbf{a}$ then we can succinctly express linear equations as:

$$\mathbf{a}^\mathsf{T}\mathbf{x} = b\,.$$

You may be wondering why we don't allow equations or inequalities with linear combinations on *both* sides of the sign, such as:

$$x_1 + 3x_2 \geq 4x_3 + 5x_4\,.$$

The reason is that we can easily rearrange any such expression to obtain an equivalent inequality or equation that has only variables on the left and constants on the right. In this example, subtracting $4x_3 + 5x_4$ from both sides gives:

$$x_1 + 3x_2 - 4x_3 - 5x_4 \geq 0\,,$$

which has the desired form, with $\mathbf{a}^\mathsf{T} = (1, 3, -4, -5)$ and $b = 0$. It will ease our definitions if we assume that all of our linear equations and inequalities have been rearranged so that there is only a constant on the right-hand side. Of course, we are free to rearrange these if it helps to make the formulation of a particular program clearer.

Given 2 vectors $\mathbf{x}$ and $\mathbf{y}$, both in $\mathbb{R}^n$, we say $\mathbf{x} = \mathbf{y}$ if and only if $x_i = y_i$ for all $1 \leq i \leq n$ (so, 2 vectors are equal if and only if all of their entries are equal; note that we have already used this convention above when considering linear combinations of vectors). Extending this idea, we will write $\mathbf{x} \leq \mathbf{y}$ to mean that $x_i \leq y_i$ for all $1 \leq i \leq n$, and adopt a similar convention for $\geq$, $<$, and $>$. Notice that one consequence of this decision is that if $\mathbf{x} \leq \mathbf{y}$ is false, we *do not necessarily have* $\mathbf{x} > \mathbf{y}$. Indeed, consider the vectors $\mathbf{x} = \begin{pmatrix}1\\0\end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix}0\\1\end{pmatrix}$. Then *none* of the following hold: $\mathbf{x} = \mathbf{y}$, $\mathbf{x} \leq \mathbf{y}$, $\mathbf{x} \geq \mathbf{y}$, $\mathbf{x} < \mathbf{y}$, or $\mathbf{x} > \mathbf{y}$. Formally, this means that the operations $(<)$ and $(>)$ give us a *partial* order over vectors—for example, the two vectors $\mathbf{x}$ and $\mathbf{y}$ defined above are incomparable with respect to this ordering.

Using this convention, we can rewrite entire systems of inequalities succinctly. For example, the set of m inequalities:

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

can be written as $A\mathbf{x} \leq \mathbf{b}$, where

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

1.2.3 Linear Independence and Rank

We say that a set of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{R}^n$ is *linearly dependent* if there is some non-trivial linear combination of $\mathbf{a}_1, \ldots, \mathbf{a}_m$ that is equal to the all zero vector. That is, $x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_m\mathbf{a}_m = \mathbf{0}$ for some set of values $x_1, \ldots, x_m$ that are not all equal to zero.

If a set of vectors is *not* linearly dependent, we say that it is *linearly independent*. Thus, $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are linearly independent if and only if $x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots x_m\mathbf{a}_m = \mathbf{0}$ holds only when $x_1 = x_2 = \cdots = x_m = 0$. We can state this more succinctly using matrices: let A is matrix whose ith column is given by $\mathbf{a}_i$. Then the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are linearly independent if and only if $A\mathbf{x} = \mathbf{0}$ has a unique solution $\mathbf{x} = \mathbf{0}$.

Note that we can regard either the columns or the rows of any matrix as vectors. Given an $m \times n$ matrix A, we can ask what is the maximum number $1 \leq r \leq n$ so that some set of r columns of A is linearly independent. We can also ask what is the maximum number $1 \leq r \leq m$ so that some set of r rows of A is linearly independent. The answer to both of these questions is always the same! This number r is called the *rank* of the matrix A: it is the maximum number of rows of A that are linearly independent and also the maximum number of columns of A that are linearly independent.

As a consequence, if $m < n$, we know that the rank of A is at most m, so any set of *more than* m columns of A must be linearly dependent. Suppose that A is a square matrix, so $m = n$. Then, if the rank of A is m, this means that all the columns and all the rows of A are linearly independent. If this is true, then for any $\mathbf{b} \in \mathbb{R}^m$, there is a single, unique solution $\mathbf{x}$ to the equation $A\mathbf{x} = \mathbf{b}$.

1.3 Linear Programs

Let's look at the programs (1.6) and (1.9) from Examples 1.1 and 1.2 in more detail. They are certainly both mathematical programs, but they have a special form. First, the decision variables represent continuous quantities (specifically, real numbers). In both cases, the objective function we are trying to optimise is a *linear combination* of the decision variables $x_1, \ldots, x_n$. Additionally, each constraint in both programs is given by a *linear inequality* over the decision variables. Whenever this is the case, we say that a mathematical program is a *linear program*.

Definition 1.4 (Linear Program)**.** A *linear program* is a mathematical program in which:

- the variables are *continuous*, *real* quantities.
- the objective is a *linear combination* of these variables, and
- each constraint is either a *linear inequality* or a *linear equation* over these variables.

We will typically refer to any particular choice of values for the decision variables of a linear program as a *solution* of that program. Note that (contrary to the usual meaning of the word solution) for us, a solution is not the best possible choice of values for the decision variables, and nor does a solution need to satisfy all the constraints of the program! We say that a solution that satisfies all the constraints of the problem is a *feasible solution* and a solution that is the best possible is an *optimal solution*. We will make these notions completely formal shortly, but first let's make things a bit easier by coming up with a standard form for writing linear programs:

Definition 1.5 (Standard Inequality Form)**.** We say that a linear program is in *standard inequality form* if it is written as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where A is an $m \times n$ matrix of real numbers, $\mathbf{c} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$.

Note that we have used the all-zero vector to concisely capture all of the non-negativity constraints in the last line. We could equivalently have written some-

thing like:

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

It is possible to transform every linear program into an equivalent program in this standard inequality form. Writing LPs in standard inequality form will be useful mainly because it makes it easy to talk about an arbitrary linear program without considering a huge number of possible cases.

Note: some textbooks will call this "canonical form," and refer to a completely different form (that we will see later) as "standard form"! In the definition, we have chosen to use the term "standard inequality form" to be as unambiguous as possible.

Let's see how an arbitrary linear program can be rewritten in this standard inequality form. There are 3 issues we need to consider:

1. We need to make sure all variables are restricted to be non-negative.
2. We need make sure our problem is a maximisation problem.
3. We need to express all of our constraints as a single vector inequality $A\mathbf{x} \leq \mathbf{b}$.

First, let us consider the sign restrictions placed on each single variable of the linear program. We will transform our linear program so that all variables must be non-negative. In general, however, we might have variables in our model that always need to be non-positive or variables that can be either positive or negative (i.e. variables that are *unrestricted*). We can handle this as follows:

- If a variable x_i is restricted to be non-positive (i.e. $x_i \leq 0$) we can introduce a new variable (call it $\bar{x}_i$ for now) that represents $-x_i$. Then, since $x_i = -\bar{x}_i$, we can replace x_i with $-\bar{x}_i$ everywhere in the program without changing its meaning. In particular, we then get the constraint that $-\bar{x}_i \leq 0$, which is equivalent to $\bar{x}_i \geq 0$, which is of the desired form.
- If a variable x_i does not appear in any restriction, then we introduce 2 new variables (call them x_i^+ and x_i^- for now). We replace x_i by $x_i^+ - x_i^-$ everywhere in our program and then add the restrictions $x_i^+ \geq 0$ and $x_i^- \geq 0$. Notice that indeed we can represent *any* number as the difference of 2 non-negative terms, so this allows any value to appear in all of the places where x_i originally was, and it requires that the *same* value appear in each place.

After carrying out the above operations, we have a new linear program in which each variable is constrained to be non-negative. Suppose that we carrying out all of the above operations we get a set of n variables. Let's re-label these as $x_1, x_2, \ldots, x_n$. Then, we can put all our decision variables together into a vectors $\mathbf{x} \in \mathbb{R}^n$.

Next, let's look at the goal of the linear program. If it is a minimisation program, we can simply negate the objective function to turn it into a maximisation problem. That is, if the first line is:

$$\min \mathbf{c}^\mathsf{T}\mathbf{x}\,,$$

we replace it with:

$$\max -\mathbf{c}^\mathsf{T}\mathbf{x}\,,$$

leaving the constraints unchanged. Again, notice that this should not affect the problem that we are solving.

Finally, we handle our constraints. We begin by changing each constraint into one or more linear inequalities of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$, where $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. To do this, we first rearrange each constraint so that all terms involving variables are on the left-hand side, and all constant terms are on the right-hand side. After doing this and simplifying, the left-hand side can be written as $\mathbf{a}^\mathsf{T}\mathbf{x}$ for some $\mathbf{a} \in \mathbb{R}^n$ and the left hand side as b for some $b \in \mathbb{R}$. Next, we examine the operator in the constraint:

- If we have $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$, we leave it as is.
- If we have $\mathbf{a}^\mathsf{T}\mathbf{x} \geq b$, we multiply both sides by -1 to obtain an equivalent inequality of the form:
$$-\mathbf{a}^\mathsf{T}\mathbf{x} \leq -b\,.$$
- If we have $\mathbf{a}^\mathsf{T}\mathbf{x} = b$, we replace it by 2 new constraints:
$$\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$$
$$-\mathbf{a}^\mathsf{T}\mathbf{x} \leq -b$$
Notice that the second inequality is equivalent to requiring that $\mathbf{a}^\mathsf{T}\mathbf{x} \geq b$. In particular, a solution $\mathbf{x}$ satisfies *both* of these inequalities if and only if $\mathbf{a}^\mathsf{T}\mathbf{x} = b$.

This process gives us a set of constraints (say, m of them), all of the same general form. That is, for each $j = 1, \ldots, m$ we have a constraint of the form[3] ${\mathbf{a}_j}^\mathsf{T}\mathbf{x} \leq b_j$. Let A be an $m \times n$ matrix, whose jth row is given by ${\mathbf{a}_j}^\mathsf{T}$, and let $\mathbf{b}$ be a $1 \times m$ vector whose jth entry is b_j. Then, we can write the entire system of constraints as:

$$A\mathbf{x} \leq \mathbf{b}\,.$$

Notice that each column of the matrix A corresponds to a decision variable and each row of A corresponds to (the left-hand side of) a constraint.

Let's see an example of how this works.

[3]Notice that $\mathbf{a}_j$ here represents the jth *vector* in our collection, and *not* the jth element of vector $\mathbf{a}$, which we would have written as a_j.

Example 1.4. Rewrite the following linear program in standard inequality form.

$$\begin{aligned} \text{minimize} \quad & 3x_1 - x_2 \\ \text{subject to} \quad & -x_1 + 6x_2 + x_3 + x_4 \geq -3 \\ & 7x_2 + x_4 = 5 \\ & x_1 + x_2 + x_3 = 1 \\ & x_3 + x_4 \leq 2 \\ & x_2, x_3 \geq 0 \\ & x_4 \leq 0 \\ & x_1 \text{ unrestricted} \end{aligned} \tag{1.11}$$

Solution. We go through the steps in order. First, let's look at the restrictions on our 4 variables. We have $x_4 \leq 0$, so we replace x_4 by $\bar{x}_4 = -x_4$ in all of our constraints to get:

$$\begin{aligned} \text{minimize} \quad & 3x_1 - x_2 \\ \text{subject to} \quad & -x_1 + 6x_2 + x_3 - \bar{x}_4 \geq -3 \\ & 7x_2 - \bar{x}_4 = 5 \\ & x_1 + x_2 + x_3 = 1 \\ & x_3 - \bar{x}_4 \leq 2 \\ & x_2, x_3, \bar{x}_4 \geq 0 \\ & x_1 \text{ unrestricted} \end{aligned}$$

Next, we see that x_1 is unrestricted. So, we replace it by the difference of two new non-negative variables, namely $x_1^+ - x_1^-$, to get:

$$\begin{aligned} \text{minimize} \quad & 3(x_1^+ - x_1^-) - x_2 \\ \text{subject to} \quad & -(x_1^+ - x_1^-) + 6x_2 + x_3 - \bar{x}_4 \geq -3 \\ & 7x_2 - \bar{x}_4 = 5 \\ & (x_1^+ - x_1^-) + x_2 + x_3 = 1 \\ & x_3 - \bar{x}_4 \leq 2 \\ & x_1^+, x_1^-, x_2, x_3, \bar{x}_4 \geq 0 \end{aligned}$$

Now, we can expand some multiplications to get:

$$\begin{aligned}
\text{minimize} \quad & 3x_1^+ - 3x_1^- - x_2 \\
\text{subject to} \quad & -x_1^+ + x_1^- + 6x_2 + x_3 - \bar{x}_4 \geq -3 \\
& 7x_2 - \bar{x}_4 = 5 \\
& x_1^+ - x_1^- + x_2 + x_3 = 1 \\
& x_3 - \bar{x}_4 \leq 2 \\
& x_1^+, x_1^-, x_2, x_3, \bar{x}_4 \geq 0
\end{aligned}$$

That takes care of our restrictions. Now, we change minimisation into maximisation by negating the objective function:

$$\begin{aligned}
\text{maximize} \quad & -3x_1^+ + 3x_1^- + x_2 \\
\text{subject to} \quad & -x_1^+ + x_1^- + 6x_2 + x_3 - \bar{x}_4 \geq -3 \\
& 7x_2 - \bar{x}_4 = 5 \\
& x_1^+ - x_1^- + x_2 + x_3 = 1 \\
& x_3 - \bar{x}_4 \leq 2 \\
& x_1^+, x_1^-, x_2, x_3, \bar{x}_4 \geq 0
\end{aligned}$$

Last, we need to handle the constraints. We first replace each equation with 2 inequalities:

$$\begin{aligned}
\text{maximize} \quad & -3x_1^+ + 3x_1^- + x_2 \\
\text{subject to} \quad & -x_1^+ + x_1^- + 6x_2 + x_3 - \bar{x}_4 \geq -3 \\
& 7x_2 - \bar{x}_4 \geq 5 \\
& 7x_2 - \bar{x}_4 \leq 5 \\
& x_1^+ - x_1^- + x_2 + x_3 \geq 1 \\
& x_1^+ - x_1^- + x_2 + x_3 \leq 1 \\
& x_3 - \bar{x}_4 \leq 2 \\
& x_1^+, x_1^-, x_2, x_3, \bar{x}_4 \geq 0
\end{aligned}$$

Finally, we multiply each side of the ($\geq$) inequalities by -1 to write them as ($\leq$)

inequalities:

$$\begin{aligned}
\text{maximize} \quad & -3x_1^+ + 3x_1^- + x_2 \\
\text{subject to} \quad & x_1^+ - x_1^- - 6x_2 - x_3 + \bar{x}_4 \leq 3 \\
& -7x_2 + \bar{x}_4 \leq -5 \\
& 7x_2 - \bar{x}_4 \leq 5 \\
& -x_1^+ + x_1^- - x_2 - x_3 \leq -1 \\
& x_1^+ - x_1^- + x_2 + x_3 \leq 1 \\
& x_3 - \bar{x}_4 \leq 2 \\
& x_1^+, x_1^-, x_2, x_3, \bar{x}_4 \geq 0
\end{aligned}$$

□

We now have a linear program in standard inequality form with 5 variables. Let's list the variables in the order as the vector $\mathbf{x}^\mathsf{T} = (x_1^+, x_1^-, x_2, x_3, \bar{x}_4)$. We can rearrange the objective and every inequality so that the variables appear in this order and make sure every variable appears in every constraint by adding some of them with 0 coefficients. This will make it easier to identify $\mathbf{c}$, A and $\mathbf{b}$. then we get:

$$\begin{aligned}
\text{maximize} \quad & -3x_1^+ + 3x_1^- + 1x_2 + 0x_3 + 0\bar{x}_4 \\
\text{subject to} \quad & x_1^+ - x_1^- - 6x_2 - x_3 + \bar{x}_4 \leq 3 \\
& 0x_1^+ + 0x_1^- - 7x_2 + 0x_3 + \bar{x}_4 \leq -5 \\
& 0x_1^+ + 0x_1^- + 7x_2 + 0x_3 - \bar{x}_4 \leq 5 \\
& -x_1^+ + x_1^- - x_2 - x_3 + 0x_4 \leq -1 \\
& x_1^+ - x_1^- + x_2 + x_3 + 0x_4 \leq 1 \\
& 0x_1^+ - 0x_1^- + 0x_2 + x_3 - \bar{x}_4 \leq 2 \\
& x_1^+, x_1^-, x_2, x_3, \bar{x}_4 \geq 0
\end{aligned}$$

So, our program in standard inequality form is:

$$\begin{aligned}
\text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\
& \mathbf{x} \geq 0
\end{aligned}$$

where:

$$\mathbf{c} = \begin{pmatrix} -3 \\ 3 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & -1 & -6 & -1 & 1 \\ 0 & 0 & -7 & 0 & 1 \\ 0 & 0 & 7 & 0 & -1 \\ -1 & 1 & -1 & -1 & 0 \\ 1 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 3 \\ -5 \\ 5 \\ -1 \\ 1 \\ 2 \end{pmatrix}$$

Notice that there is some ambiguity here, since in general we might get different $\mathbf{c}$, $\mathbf{b}$ and A if we decide to list our variables in a different order in $\mathbf{x}$. Of course, any order would be okay, since it just amounts to choosing what order we write our linear combinations and inequalities in. However, to make things clearer, from now on we will always order our variables as follows: We form $\mathbf{x}$ so that variables are listed, top to bottom, in *alphabetical* order. In this ordering, we treat suppose that x_i before x_j whenever $i < j$, $\bar{x}_i$ is treated exactly like x_i (notice we will never have both x_i and $\bar{x}_i$ in the same program), and x_i^+ comes before x_i^-. Notice that the ordering we just used in our example satisfies these properties.

Note: We did not have time to cover the material below in week 1, so will cover it in week 2.

Consider now an arbitrary LP, and suppose that we have rewritten it into our standard inequality form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

Recall that $\mathbf{x}$ is our vector of decision variables. Any choice of values for the decision variables (i.e. any specific choice of $\mathbf{x}$) is called a solution of the linear program (rather confusingly). What we are actually interested in is finding feasible solutions and optimal solutions as defined below.

Definition 1.6 (Feasible Solution of a Linear Program)**.** A solution $\mathbf{x} \in \mathbb{R}^n$ is a *feasible solution* to a linear program (in standard inequality form) if and only if $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$.

Note that here we saved ourselves a lot of tedious cases (one for each possible type of constraint) by placing the program in standard inequality form

Definition 1.7 (Optimal Solution of a Linear Program)**.** We say that a feasible solution $\mathbf{x}$ is an *optimal solution* to a linear program in standard inequality form if and only if $\mathbf{c}^\mathsf{T}\mathbf{x} \geq \mathbf{c}^\mathsf{T}\mathbf{x}'$ for any other feasible solution $\mathbf{x}'$ of this program.

Here we assume that the program is in standard inequality form, and so is a maximisation problem. We could also define optimal solutions $\mathbf{x}$ for minimisation problems directly, in which case we should have $\mathbf{c}^\mathsf{T}\mathbf{x} \leq \mathbf{c}^\mathsf{T}\mathbf{x}'$ for any other feasible solution $\mathbf{x}'$ of this program. Intuitively, all of the above definitions just say that a feasible solution is optimal if and only if there is no other feasible solution that is "better" than it, where we measure "better" according to the problem's objective and goal. Notice that it is easy to determine if a given solution is feasible: we just check if it satisfies each constraint. One of the main questions we will concern ourselves with is how to determine if a given solution is *optimal* and how to find such a solution.

Week 2

Modelling With Linear Programs

Here, we will discuss the notions of *feasibility* and *optimality* in linear programming, and start modelling some common kinds of problems using linear programs.

Note: We did not have time to cover the material below in week 1, so we covered it in week 2.

Consider now an arbitrary LP in n variables. Informally, a *solution* to our LP is (somewhat confusingly) any assignment of real values to the variables. A *feasible solution* to our LP is any solution that satisfies all the constraints and sign restrictions in our LP. An *optimal solution* to the LP is any feasible solution that achieves the goal of the LP.

In order to give the more formal definitions, assume that we have rewritten our LP into our standard inequality form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

Recall that $\mathbf{x}$ is our vector of variables. Any choice of values for the variables (i.e. any specific choice of $\mathbf{x}$) is called a solution of the linear program (rather confusingly). What we are actually interested in is finding feasible solutions and optimal solutions as defined below.

Definition 2.1 (Feasible Solution of a Linear Program)**.** A solution $\mathbf{x} \in \mathbb{R}^n$ is a *feasible solution* to a linear program (in standard inequality form) if and only if $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$.

Note that here we saved ourselves a lot of tedious cases (one for each possible type of constraint) by placing the program in standard inequality form

Definition 2.2 (Optimal Solution of a Linear Program)**.** We say that a feasible solution $\mathbf{x}$ is an *optimal solution* to a linear program in standard inequality form if and only if $\mathbf{c}^\mathsf{T}\mathbf{x} \geq \mathbf{c}^\mathsf{T}\mathbf{x}'$ for any other feasible solution $\mathbf{x}'$ of this program.

Here we assume that the program is in standard inequality form, and so is a maximisation problem. We could also define optimal solutions $\mathbf{x}$ for minimisation problems directly, in which case we should have $\mathbf{c}^\mathsf{T}\mathbf{x} \leq \mathbf{c}^\mathsf{T}\mathbf{x}'$ for any other feasible solution $\mathbf{x}'$ of this program. Intuitively, all of the above definitions just say that a feasible solution is optimal if and only if there is no other feasible solution that is "better" than it, where we measure "better" according to the problem's objective and goal. Notice that it is easy to determine if a given solution is feasible: we just check if it satisfies each constraint. One of the main questions we will concern ourselves with is how to determine if a given solution is *optimal* and how to find such a solution.

2.1 Production Problems

We have already seen some examples of *mixture* problems that can be formulated as linear programs. There, our decision variables represented the amounts of various ingredients to include. Now, we discuss more complex *production problems* that can also be modelled as linear programs. In these problems, we have a set of different *processes*, each converting input materials into output products. The goal is to do this in the best way possible. Depending on the context, we might want to either:

- Maximise the value of the products we produce, given constraints on number of resources available
- Minimise the cost of the resources we use, given constraints on what we must produce

We can also model various combinations of these two approaches by, for example, considering the *profits* or values minus costs as our objective.

For these problems, it is best to introduce one decision variable for each *process* or *activity*. The value of this variable will tell us the *level* or *extent* to which we make use of this process in our production. Exactly what this means will depend on the exact problem considered and the units chosen to measure things. Let's see an example to clarify things:

Example 2.1. A factory makes 2 different parts (say, part X and part Y). Their plant has 4 separate processes in place: there are two older processes (say, process 1 and 2) that produce parts X and Y directly, as well as two different integrated processes for producing both X and Y simultaneously. The 4 processes can be run simultaneously, but require labour, raw metal, and electricity. The hourly inputs and outputs for each process are as follows:

	Outputs		Inputs		
Process	X	Y	Metal	Electricity	Labour
1	4	0	100 kg	800 kWh	16 hrs
2	0	1	70 kg	600 kWh	16 hrs
3	3	1	120 kg	2000 kWh	50 hrs
4	6	3	270 kg	4000 kWh	48 hrs

In a typical day, the plant has an available stock of 6000 kg of metal, and the has budgeted 100000 kWh of power usage, 1000 hours of labour. Suppose that each part X sells for £1000 and each part Y sells for £1800. How should production be scheduled to maximise daily revenue?

To model this as a linear program, we need to assume that all quantities are continuous. If this schedule is going to be implemented over a long period, this is reasonable: if tomorrow we can "pick up" production of a partial X or Y part where we left off, we might as well count the fraction of each such part produced into a typical current day's revenue.

We want to maximise revenue in this question, which will ultimately involve deciding how many parts X and Y to make in a day. However, notice that these are not the only decisions we need to make! We need to know *how* to produce them, as well. Thinking more carefully, we see that the *activities* here are Processes 1,2,3, and 4, and we need to decide how many hours (or fractions thereof) we run each one for. Once we know this, we will actually know how many X and Y we produce. So, let's introduce 4 variables, p_1, p_2, p_3, and p_4, to represent how many hours we run the respective processes for. Immediately, we see that these variables must be non-negative.

Now, let's see if we can formulate the objective. Looking at process 1, for example, we see that running for a *single* hour produces 4 X and 0 Y, which gives revenue $4 \cdot 1000 + 0 \cdot 1800$. Thus, if we execute it for p_1 hours, we will have total revenue $4000p_1$. Doing this for each process and adding the results, we get:

$$\begin{aligned}(4 \cdot 1000)p_1 + (1 \cdot 1800)p_2 + (3 \cdot 1000 + 1 \cdot 1800)p_3 + (6 \cdot 1000 + 3 \cdot 1800)p_4 \\ = 4000p_1 + 1800p_2 + 4800p_3 + 11400p_4\end{aligned}$$

This, then is our objective function, which we want to maximise.

Now, we need to incorporate our resource constraints. Again, we can look at the resources needed to run process 1 for a *single* hour requires 100kg metal. So, if we run for p_1 hours, we will consume $100p_1$ kg of metal. We can do the same thing for each process and add up the results to find that the total metal consumed will be:

$$100p_1 + 70p_2 + 120p_3 + 270p_4$$

We only have 6000kg of metal on hand for a day so we need to add a constraint, ensuring that:

$$100p_1 + 70p_2 + 120p_3 + 270p_4 \leq 6000\,.$$

If we do the same thing for electricity and labour we get the additional constraints:

$$\begin{aligned} 800p_1 + 600p_2 + 2000p_3 + 4000p_4 &\leq 100000 \\ 16p_1 + 16p_2 + 50p_3 + 48p_4 &\leq 1000 \end{aligned}$$

Altogether, we get the following linear program:

$$\begin{aligned} \text{maximize} \quad & 4000p_1 + 1800p_2 + 4800p_3 + 11400p_4 \\ \text{subject to} \quad & 100p_1 + 70p_2 + 120p_3 + 270p_4 \leq 6000 \\ & 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \leq 100000 \\ & 16p_1 + 16p_2 + 50p_3 + 48p_4 \leq 1000 \\ & p_i \geq 0, \quad \text{for each } i = 1, 2, 3, 4 \end{aligned}$$

If we wanted to know how many of X and Y the optimal schedule produced, we could of course use our table together with p_1, p_2, p_3, and p_4 to calculate the answer. Another approach (that will be useful shortly) is to model the input-output relationship explicitly. Let's introduce 2 new variables x and y, which represent how much of X and Y, respectively, we will produce and 3 new variables m, e, l representing how much metal, electricity and labour we consume. Then, we have:

$$\begin{aligned} x &= 4p_1 + 3p_3 + 6p_4 \\ y &= p_2 + p_3 + 3p_4 \\ m &= 100p_1 + 70p_2 + 120p_3 + 270p_4 \\ e &= 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \\ l &= 16p_1 + 16p_2 + 50p_3 + 48p_4 \end{aligned}$$

We can add these constraints to the linear program, and then reformulate our program easily as:

$$\begin{aligned}
\text{maximize} \quad & 1000x + 1800y \\
\text{subject to} \quad & x = 4p_1 + 3p_3 + 6p_4 \\
& y = p_2 + p_3 + 3p_4 \\
& m = 100p_1 + 70p_2 + 120p_3 + 270p_4 \\
& e = 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \\
& l = 16p_1 + 16p_2 + 50p_3 + 48p_4 \\
& m \leq 6000 \\
& e \leq 100000 \\
& l \leq 1000 \\
& p_1, p_2, p_3, p_4 \geq 0 \\
& x, y \text{ unrestricted} \\
& m, e, l \text{ unrestricted}
\end{aligned} \tag{2.1}$$

Note that this makes it a lot easier to see what's going on. However, we have more than doubled the number of variables in the linear program! However, notice that all we've really done is introduce "names" or "shorthands" for the quantities we are interested in and defined them using equations.

You may be wondering why we declared x, y, m, e, l as unrestricted variables. We did this to make sure we weren't introducing any additional constraints when we introduced our names. For example, if we declared that variable $x \geq 0$ then (since $x = 4p_1+3p_3+6p_4$) we would be adding a constraint that $4p_1+3p_3+6p_4 \geq 0$. It could be complicated to determine whether or not this changes the program's optimal value, so the safest thing to do is to leave all of our "shorthand" variables unrestricted. Furthermore, we will see in later lectures that unrestricted variables in equations can be eliminated before solving a program, so leaving these variables as unrestricted will allow us (or a computer) to eliminate them from the problem before solving it.

Let's look at an example of a production problems from the other direction:

Example 2.2. Suppose that our factory in Example 2.1 wants to determine its daily operating budget. It has determined that there is daily demand for 120 parts X and 50 parts Y. Suppose now that there is an unlimited amount of metal, electricity, and labour available, but the cost of metal is £5 per kg, the cost of electricity is £0.15 per kWh, and the cost of labour is £20 per hour. How can it schedule production to meet its demand as cheaply as possible?

In this problem, our decision variables will remain the same, since we are still trying to work out a schedule of activities. However, now the amount of each part produced gives us a *constraint* and the raw materials lead us to the *objective*. Let's adopt an explicit input-output approach from before. Then, we will have the same equations relating the i, o and x variables. We need to formulate our new objective. It is simply:

$$5m + 0.15e + 20l$$

Moreover, our constraints on the number of X and Y produced give us 2 inequalities for x and y:

$$x \geq 120 \qquad\qquad y \geq 50$$

Thus, our final program is given by:

$$\begin{aligned}
\text{minimize} \quad & 5m + 0.15e + 20l \\
\text{subject to} \quad & x = 4p_1 + 3p_3 + 6p_4 \\
& y = p_2 + p_3 + 3p_4 \\
& m = 100p_1 + 70p_2 + 120p_3 + 270p_4 \\
& e = 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \\
& l = 16p_1 + 16p_2 + 50p_3 + 48p_4 \\
& x \geq 120 \\
& y \geq 50 \\
& p_1, p_2, p_3, p_4 \geq 0 \\
& x, y \text{ unrestricted} \\
& m, e, l \text{ unrestricted}
\end{aligned} \tag{2.2}$$

Using a similar method we can actually perform both of these optimisations to determine how to maximise *profits*, which are simply given by revenue minus costs.

Example 2.3. Suppose that our factory in the previous 2 examples now wants to find a production schedule that maximises its daily *profits* defined as revenue minus costs. How can this be done? You should assume that any amount of resources are available, and that any number of parts can be sold (where the prices are given as in the previous 2 examples).

We have the same equations relating inputs to outputs as in the previous example. Now, however, we will have no constraints on inputs or outputs. We introduce

positive terms in the objective for revenue, as in Example 2.1 and negative terms for costs, which are calculated as in Example 2.2. Altogether, we get:

$$\begin{aligned}
\text{maximize} \quad & 1000x + 1800y - 5m - 0.15e - 20l \\
\text{subject to} \quad & x = 4p_1 + 3p_3 + 6p_4 \\
& y = p_2 + p_3 + 3p_4 \\
& m = 100p_1 + 70p_2 + 120p_3 + 270p_4 \\
& e = 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \\
& l = 16p_1 + 16p_2 + 50p_3 + 48p_4 \\
& p_1, p_2, p_3, p_4 \geq 0 \\
& x, y \text{ unrestricted} \\
& m, e, l \text{ unrestricted}
\end{aligned}$$

Finally, let's see an example of a more complicated situation:

Example 2.4. A medical testing company is making diagnostic tests. Each test requires a combination of 3 different reagents:

Test	Reagents Needed		
	1	2	3
Standard	0.9 ml	1.2 ml	
Rapid	1.5 ml		1.0 ml

Each reagent can be synthesised from a combination of more basic chemicals (let's call them chemical A, B, and C), which requires some amount of laboratory time. The relevant materials and costs are summarised in the following table:

Reagent	Chemicals Needed			Lab time	Price
	A	B	C	to synthesise	
1	1.0 ml	0.3 ml	1.5 ml	0.02 hrs/ml	£2.40/ml
2	0.5 ml	0.2 ml	1.0 ml	0.04 hrs/ml	£1.60/ml
3	0.2 ml	1.8 ml	0.6 ml	0.05 hrs/ml	£1.50/ml

The company has taken on a contract to produce 1000 standard tests and 2300 rapid tests. It has 100 hours of laboratory time available at a cost of £150 per hour, 1100ml of chemical A, 1250ml of chemical B, and 1800ml of chemical C available. Additionally, it can purchase and sell an unlimited amount of each reagent for the specified price. Find a production plan that fulfils the contract at the lowest net cost, taking into account any money recovered by the sale of excess reagents.

Solution. This is a complex *multi-stage* production problem, but the same general approach we used before works. As our first step, we should think about what decisions need to be made. Here, a production schedule must decide how much of reagents 1, 2, and 3 to synthesise and how much to purchase or sell. Let's focus first on how much of each reagent to produce: we let r_1, r_2, r_3 be a variable representing how many ml of each to produce. Then, $r_1, r_2, r_3 \geq 0$. Given values for r_1, r_2, r_3, we can work out how much of each chemical will be required, and how much lab time will be needed. In order to make things easier to read, we can introduce unrestricted variables a, b, c, and l to name these quantities. Then, we

have:

$$\begin{aligned} a &= 1.0r_1 + 0.5r_2 + 0.2r_3 \\ b &= 0.3r_1 + 0.2r_2 + 1.8r_3 \\ c &= 1.5r_1 + 1.0r_2 + 0.6r_3 \\ l &= 0.02r_1 + 0.04r_2 + 0.05r_3 \end{aligned}$$

We also know that we must produce 1000 standard tests, which will require $1000 \cdot 0.9$ ml of reagent 1 and $1000 \cdot 1.2$ ml of reagent 2, and 2300 rapid tests, requiring $2300 \cdot 1.5$ ml of reagent 1 and $2300 \cdot 1.0$ ml of reagent 3. If we produce less than the required amount of any reagent, we must pay a cost to purchase the difference, and if we produce more than the required amount, we can sell the excess. Thus, the total contribution to our net costs can be written as:

$$2.40((1000 \cdot 0.9 + 2300 \cdot 1.5) - r_1) + 1.60(1000 \cdot 1.2 - r_2) + 1.50(2300 \cdot 1.0 - r_3)$$

Our laboratory time also factors into the cost as $150l$, and we have constraints on the total amount of laboratory time and each chemical available. Putting everything together, we have:

$$\begin{aligned} \text{minimize} \quad & 150l + 2.40((1000 \cdot 0.9 + 2300 \cdot 1.5) - r_1) \\ & + 1.60(1000 \cdot 1.2 - r_2) + 1.50(2300 \cdot 1.0 - r_3) \\ \text{subject to} \quad & a = 1.0r_1 + 0.5r_2 + 0.2r_3 \\ & b = 0.3r_1 + 0.2r_2 + 1.8r_3 \\ & c = 1.5r_1 + 1.0r_2 + 0.6r_3 \\ & l = 0.02r_1 + 0.04r_2 + 0.05r_3 \\ & a \leq 1100 \\ & b \leq 1250 \\ & c \leq 1800 \\ & l \leq 100 \\ & r_1, r_2, r_3 \geq 0 \\ & a, b, c, l \text{ unrestricted} \end{aligned}$$

If desired, we could eliminate a, b, c, and l (by substituting the right-hand side of

the constraint equation for each of them) to arrive at:

$$\begin{aligned}
\text{minimize} \quad & 150(0.02r_1 + 0.04r_2 + 0.05r_3) + 2.40((1000 \cdot 0.9 + 2300 \cdot 1.5) - r_1) \\
& + 1.60(1000 \cdot 1.2 - r_2) + 1.50(2300 \cdot 1.0 - r_3) \\
\text{subject to} \quad & 1.0r_1 + 0.5r_2 + 0.2r_3 \leq 1100 \\
& 0.3r_1 + 0.2r_2 + 1.8r_3 \leq 1250 \\
& 1.5r_1 + 1.0r_2 + 0.6r_3 \leq 1800 \\
& 0.02r_1 + 0.04r_2 + 0.05r_3 \leq 100 \\
& r_1, r_2, r_3 \geq 0
\end{aligned}$$

Note that this type of simplification is only valid because each of these variables is *unrestricted*. Otherwise, we would also have to make sure that the corresponding right-hand sides remained non-negative or non-positive (according to the restrictions on the left-hand side).

Going the other direction, we could introduce additional variables to make the objective easier to read. For example, we could let s_1 be the amount of reagent 1 that we have to purchase or sell. Then, we would have a constraint:

$$r_1 + s_1 = 1000 \cdot 0.9 + 2300 \cdot 1.5$$

saying that the total amount of reagent 1 produced together with that purchased or sold must exactly equal the amount required to make the tests. Then, we can replace the term $2.40((1000 \cdot 0.9 + 2300 \cdot 1.5) - r_1)$ in the objective with $2.40s_1$. We could do this for each of the reagents separately, introducing s_1, s_2, and s_3. Notice that here we again need to be careful to make s_1 unrestricted—if we say that $s_1, s_2, s_3 \geq 0$ we will introduce an extra constraint into the problem saying that we can only *purchase* reagents. □

2.2 Transportation Problems

Next, we consider another class of problems that we can solve with linear programming, called *transportation problems*. These involve moving or assigning goods, materials, or resources from one set of locations to another.

Example 2.5. A mining company has 2 mines, where ore is extracted, and 3 warehouses, where ore is stored. Currently, there is 45Mg of ore divided amongst the mining locations. In order to prepare it for sale, this ore needs to be distributed to the warehouses. The amount of ore available at each mine, and the amount of ore required at each warehouse is as follows:

	Ore Available
Mine 1	19
Mine 2	26

	Ore Required
Warehouse 1	14
Warehouse 2	11
Warehouse 3	20

Due to different distances and shipping methods, the cost (in thousands of pounds) to ship 1 Mg depends on where it is being shipped from and where it is being shipped to, as follows:

	Warehouse 1	Warehouse 2	Warehouse 3
Mine 1	10	5	12
Mine 2	9	7	13

Suppose that these costs scale up linearly in the amount of ore that is shipped (for example, it costs $3 \cdot 10$ to ship 3Mg of ore from Mine 1 to Warehouse 1. How should we send the ore from the mines to the warehouses to minimise the overall transportation cost?

In this example, our decision variables should allow us to decide *how much ore* is routed between each mine and each warehouse. We can think of sending a shipment from Mine i to Warehouse j as an *activity*, and then, as usual, introduce a decision variable to describe the extent to which this activity is carried out. That is, we will introduce a variable $x_{i,j}$ for each Mine $i = 1, 2$ and each Warehouse $j = 1, 2, 3$ representing how many Mg of ore we send from Mine i to Warehouse j. It doesn't make sense for this to be negative in our setting (since we can't ship less than 0 Mg). Altogether, we will then have 6 non-negative decision variables $x_{1,1}, x_{1,2}, x_{1,3}, x_{2,1}, x_{2,2}, x_{2,3}$ (note that we could also have named them $x_1, \ldots, x_6$, but using a pair of numbers as an index will help us remember what they represent). The total cost of a proposed shipping schedule will be given by multiplying each $x_{i,j}$ by the appropriate entry in the table for costs. For example, we will incur a total cost of $5x_{1,2}$ due to shipping $x_{1,2}$ Mg of ore from Mine 1 to Warehouse 2. Combining all of these costs, we obtain the following as the total cost:

$$10x_{1,1} + 5x_{1,2} + 12x_{1,3} + 9x_{2,1} + 7x_{2,2} + 13x_{2,3}$$

What should our constraints be? We know that there are only 19Mg of ore available at Mine 1, so the total amount of ore sent from this mine should be at

most 19Mg:

$$x_{1,1} + x_{1,2} + x_{1,3} \leq 19\,,$$

or, more succinctly,

$$\sum_{j=1}^{3} x_{1,j} \leq 19. \tag{2.3}$$

We get a similar constraint for every mine.

Similarly, we know that Warehouse 1 must receive 14Mg of ore, from all of the shipments it receives, so we should have:

$$\sum_{i=1}^{2} x_{i,1} = x_{1,1} + x_{2,1} \geq 14\,. \tag{2.4}$$

We get a similar constraint for every warehouse.

Altogether, our program looks like:

$$\begin{aligned}
\text{minimize} \quad & 10x_{1,1} + 5x_{1,2} + 12x_{1,3} + 9x_{2,1} + 7x_{2,2} + 13x_{2,3} \\
\text{subject to} \quad & \sum_{j=1}^{3} x_{1,j} \leq 19 \\
& \sum_{j=1}^{3} x_{2,j} \leq 26 \\
& \sum_{i=1}^{2} x_{i,1} \geq 14 \\
& \sum_{i=1}^{2} x_{i,2} \geq 11 \\
& \sum_{i=1}^{2} x_{i,3} \geq 20
\end{aligned}$$

In general, a transportation problem involves shipping some amounts *available* at one set of locations i to satisfy some *needs* at another set of locations j. We will get one constraint for each *source* location i, of the form:

$$\sum_{j} x_{i,j} \leq s_i$$

where s_i is the *supply* available at location i. Similarly, we will get one constraint for each *destination* location j of the form:

$$\sum_{i} x_{i,j} \geq d_j$$

where d_j is the *demand* at the location j. In our example, the mines were sources, each with a supply s_i representing how much ore was available, and the warehouses were destinations, each with a demand d_j representing how much ore was required.

Note that if the total supply at all source locations is *less* than the total demand at all destinations, we cannot expect to solve the problem at all. This leads us to the notion of *infeasible* linear programs, which we will discuss shortly.

2.3 Integer vs Fractional Solutions

Finally, let us discuss an important implication of Definition 1.4 for modelling problems. The first property means that we need to be careful when modelling or interpreting some problems as linear programs. In the diet problem (Example 1.1) we had to assume that it was okay to purchase and consume any *fraction* of a single serving and, indeed, this is what happens in the optimal solution. However, if a problem involves inherently discrete quantities (say, cars or people) the answer we get may not make sense: for example, what does it mean to sell 1.4 cars or to hire 0.3 people? The following example illustrates some of related issues that can arise:

Example 2.6. You are assembling an investment portfolio. You have £10000 to invest and want to divide it between the following options: there are 10 shares available in Company A, each costing £500 and 1 share available in Company B, costing £10000. Your financial modelling predicts that in 5 years, each share of company A will be worth £700, and each share of Company B will be worth £13000. Which shares should you buy to maximise your predicted profit?

Let's let a be the number of shares in Company A that we purchase and b be the number of shares in Company B that we purchase. Then, we have the following optimisation problem:

$$\begin{aligned} \text{maximize} \quad & 200a + 3000b \\ \text{subject to} \quad & 500a + 10000b \leq 10000 \\ & a \leq 10 \\ & b \leq 1 \\ & a, b \geq 0 \end{aligned} \tag{2.5}$$

Here, our objective and our constraints are *linear*, so if we let $a, b \in \mathbb{R}$, then this is a linear program, with solution $a = 10, b = 0.5$ with value $200 \cdot 10 + 3000 \cdot 0.5 =$

3500. However, if we can't buy *part* of a share in company B then this doesn't make sense. In that case, we want to consider only solutions $a, b \in \mathbb{Z}$. We could look at our previous solution and just round everything down (notice this guarantees we will stay feasible). We set $a = 10, b = 0$, which gives us a profit of 2000. But, if we set $a = 0, b = 1$, we get a profit of 3000. This the optimal (i.e. the best possible) *integer* solution to the problem. Notice that this is structurally quite different than the optimal *fractional* solution, and also worth a lot more than our naive rounded solution!

In general, obtaining an *integer* solution to a linear program is a very difficult problem, which is beyond the scope of this module. For the remainder of this module, we will largely ignore issues of integrality, and assume that a fractional solution to a problem makes sense. In practice it is important to keep them in mind.

Week 3

Geometry of Linear Programming

So far, we have thought of linear programs as algebraic objects that we can manipulate to obtain a standard form, or as tools for modelling problems. Now, we will see that they can also be viewed *goemetrically*. We will focus on intuition for 2 dimensions first, since this is easy to draw. We will look at programs with 2 variables, and use one axis for each variable. Note that if we consider the vector $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ starting from the origin, then it's endpoint is (x_1, x_2). Thus, we can treat $\mathbf{x}$ as either a vector (that is, a direction and magnitude starting from the origin) or a point in 2 dimensional Euclidean space.

We know that any linear program in n variables $(x_1, \ldots, x_n) = \mathbf{x}^\mathsf{T}$ can be written as a maximisation problem with an objective function of the form $\mathbf{c}^\mathsf{T}\mathbf{x}$ and a set of constraints, each of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$, where $\mathbf{c}, \mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. So, the objective and the left-hand side of each constraints is a *linear function* $f(\mathbf{x}) = a_1x_1 + a_2x_2 + \cdots + a_nx_n$ of the variables. Just as with functions of 2 variables in Calculus 2, we can find the direction in which this function is increasing most rapidly by considering its *gradient*, which here will be a vector:

$$(\tfrac{\partial f}{dx_1}, \tfrac{\partial f}{dx_2}, \ldots, \tfrac{\partial f}{dx_n})$$

whose ith component is the partial derivative of$f$ with respect to x_i. Note that since f is a linear combination, these partial derivatives are easy to compute—we have:

$$\frac{df}{\partial x_i} = a_i,$$

for every i. Thus, our gradient at any point $\mathbf{x}$ will always be given by $\mathbf{a}^\mathsf{T}$ and so at any point $\mathbf{x}$ in space, the function $f(\mathbf{x}) = \mathbf{a}^\mathsf{T}\mathbf{x}$ is increasing the most rapidly in the direction $\mathbf{a}$.

We can also ask for the *level curves* of such a combination—given some fixed value z, what does the set of all solutions $\mathbf{x}$ that have objective value z look like?

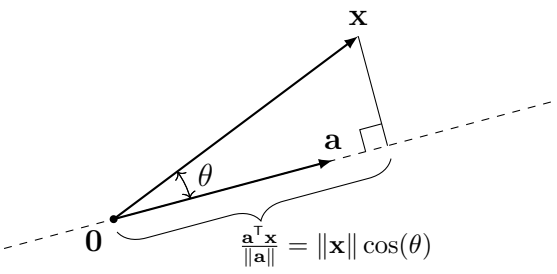
For any value z, this set will be given by a curve (or, in higher dimensions, a surface) that has a tangent *perpendicular* to the gradient that every point. Since our gradient is always $\mathbf{c}$ at all points, our level curves will always simply be the set of all points at right angles to $\mathbf{c}$. In 2 dimensions, this is just a line perpendicular to $\mathbf{c}$ and in 3 dimensions, this is a plane that is perpendicular to $\mathbf{c}$. In higher dimensions, we call this set a *hyperplane*. In all cases call $\mathbf{c}$ the *normal vector* of this line (or plane, or hyperplane).

To summarise, the set of all points attaining an objective value of z (that is, satisfying the equation $\mathbf{c}^\mathsf{T}\mathbf{x} = z$), is just a set of points that lies on a line or plane lying at a right angle to a normal vector in the direction $\mathbf{c}$. The value of z will govern how far away from the origin this line or plane is: the larger z is, the farther from the origin we have to move the plane.

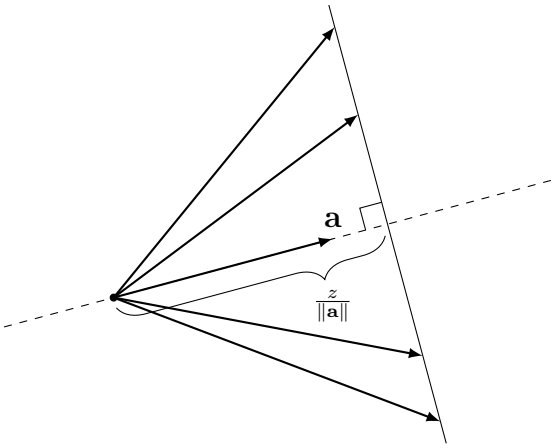
We can also see this without using calculus. Recall that the *dot product* of 2 vectors $\mathbf{a}$ and $\mathbf{x}$ in n-dimensional Euclidean space satisfies:

$$\mathbf{a}^\mathsf{T}\mathbf{x} = a_1x_1 + \cdots + a_nx_n = \|\mathbf{a}\|\|\mathbf{x}\|\cos(\theta)$$

where θ is the angle between the vectors $\mathbf{a}$ and $\mathbf{x}$. If we divide both sides of this equation by $\|\mathbf{a}\|$, and recall that the length of any side of a right triangle is given by the hypotenuse times the cosine of the adjacent angle, we can draw a picture that looks like this:



A solution of $\mathbf{a}^\mathsf{T}\mathbf{x} = z$ is then a vector $\mathbf{x}$ that together with the origin, forms a right triangle with one side of length $z/\|\mathbf{a}\|$ lying along the vector $\mathbf{a}$. Again, we can think of $\mathbf{a}$ as giving us a *direction* in space, and then the set of points with $\mathbf{a}^\mathsf{T}\mathbf{x} = z$ can all be obtained by walking exactly the distance $z/\|\mathbf{a}\|$ from the origin in the direction given by the vector $\mathbf{a}$, and then *any distance* in perpendicular to $\mathbf{a}$, as shown below:



Again, we find that the set of level curves of $\mathbf{a}^\mathsf{T}\mathbf{x}$ will always be given by a line perpendicular to $\mathbf{a}$. This line crosses the ray passing through $\mathbf{a}$ at some point that depends on how big z is: as z grows larger, we need to move our line further in the direction of $\mathbf{a}$ to make sure the labelled distance is $z/\|\mathbf{a}\|$. Note that if $z < 0$, then we simply need to walk in the *opposite direction* pointed to by $\mathbf{a}$ from the origin.

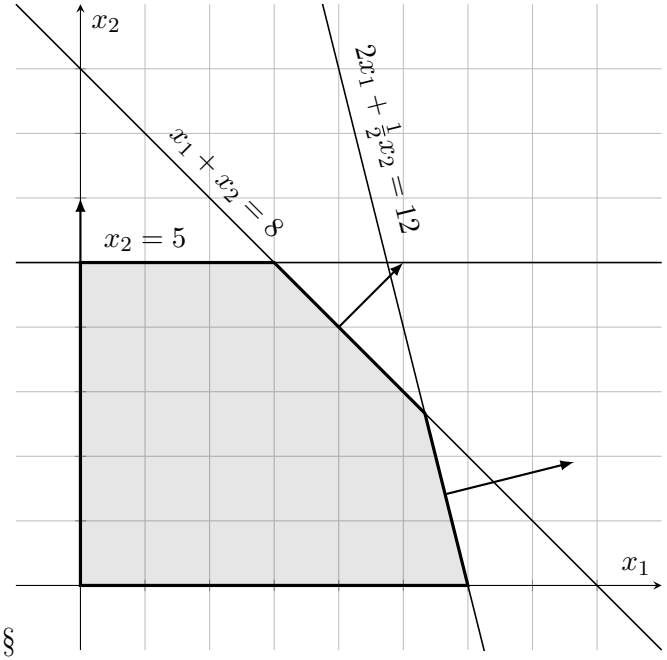
Returning now to linear programs, consider the set of solutions that satisfy a constraint of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$. The boundary of this region is simply a line of the form $\mathbf{a}^\mathsf{T}\mathbf{x} = b$. As we have seen this is a line perpendicular to $\mathbf{a}$. The region contains this boundary as well as all solutions with $\mathbf{a}^\mathsf{T}\mathbf{x} < b$, which will lie on one side of the boundary. In order to figure out which side they lie on, we simply consider the normal vector $\mathbf{a}$ of the boundary. As we have seen, this vector points in the direction in which $\mathbf{a}^\mathsf{T}\mathbf{x}$ is *increasing*. Thus, if we choose any solution on the line and draw the a vector starting at this solution and pointing in the direction of $\mathbf{a}$, then all solutions $\mathbf{x}$ on the same side of the line $\mathbf{a}^\mathsf{T}\mathbf{x} = b$ as this vector will have $\mathbf{a}^\mathsf{T}\mathbf{x}$ *larger* than b. It follows that all $\mathbf{x}$ with $\mathbf{a}^\mathsf{T}\mathbf{x} < b$ then lie on the side of the line *opposite* this vector.

Altogether each constraint $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$ gives us a *half-plane*, with a boundary given by the line $\mathbf{a}^\mathsf{T}\mathbf{x} = b$. Our linear program will have several such constraints, and our feasible solutions must satisfy *all* of them at once. So, what we really want is a solution lying in the *common intersection* of all of the half-planes defined by the constraints, as well as our sign restrictions.

Using this we can easily sketch the region of linear programs with 2 variables. As an example, suppose we want to sketch the set of feasible solutions to a linear program given by:

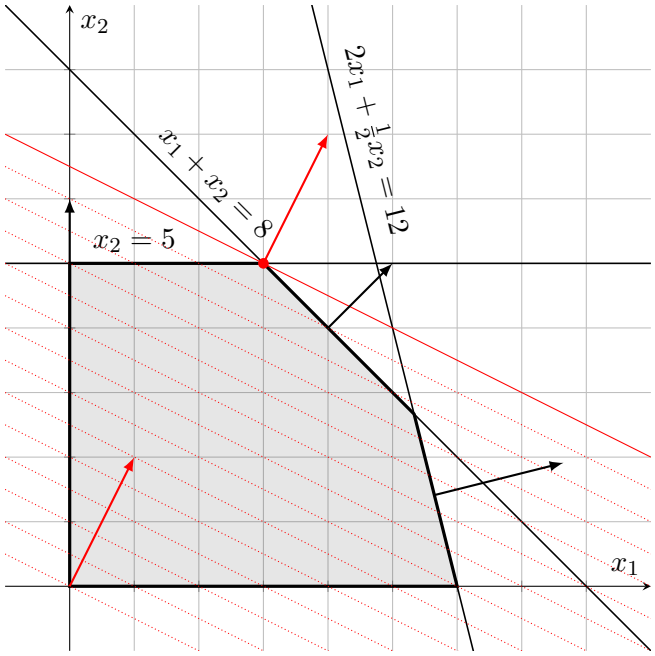
$$\begin{aligned} \text{maximize} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & x_2 \leq 5 \\ & x_1 + x_2 \leq 8 \\ & 2x_1 + \frac{1}{2}x_2 \leq 12 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{3.1}$$

If we sketch the constraints and shade the feasible region, we get something like this:



§§

In our diagram, for each constraint of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$, we have drawn the boundary given by the associated equation $\mathbf{a}^\mathsf{T}\mathbf{x} = b$, and we have also drawn a normal vector for each boundary corresponding to $\mathbf{a}$: the first constraint has $\mathbf{a}^\mathsf{T} = (0, 1)$, the next has $\mathbf{a}^\mathsf{T} = (1, 1)$, and the last has $\mathbf{a}^\mathsf{T} = (2, \frac{1}{2})$. This vector shows us in which direction the left hand side of the corresponding constraint is *increasing*, and so we shade the opposite side of the line as the feasible region. Finally, notice that we also must remember to included the restrictions $x_1 \geq 0$ and $x_2 \geq 0$, which give the left and lower boundaries of the feasible region, respectively.



Our goal is then to find a solution in the shaded region that makes the dot-product $\mathbf{c}^\mathsf{T}\mathbf{x}$ as large as possible. This is again, just a linear combination, so we know that for any constant z, the set of all solutions for which $\mathbf{c}^\mathsf{T}\mathbf{x} = z$ will be a line perpendicular to $\mathbf{c}$. Recalling the definition of an optimal solution, $\mathbf{x}$ is optimal if and only if $\mathbf{x}$ lies in the feasible region and no other feasible solution $\mathbf{x}'$ has $\mathbf{c}^\mathsf{T}\mathbf{x}' \geq \mathbf{c}^\mathsf{T}\mathbf{x}$. In terms of the *value* z of the objective function, we want to find a line perpendicular to $\mathbf{c}$ that intersects at least one solution of the feasible region so that no other part of the feasible region lies on the side of this line corresponding $\mathbf{c}$—any such solution $\mathbf{x}'$ would correspond to a feasible solution making $\mathbf{c}^\mathsf{T}\mathbf{x}'$ larger. Graphically, we imagine increasing our value z slowly, sweeping our perpendicular line along in the direction that $\mathbf{c}$ solutions. We continue as long as this line intersects the feasible region in at least 1 solution. We stop when sweeping the line any further out causes it to no longer intersect the feasible region. Then, any solution that lies the line w

The picture above shows the vector $\mathbf{c}$, together with dotted red lines corresponding to those solutions $\mathbf{x}$ with $\mathbf{c}^\mathsf{T}\mathbf{x} = 1, 2, \ldots, 13$.[1] Once we reach the line $\mathbf{c}^\mathsf{T}\mathbf{x} = 13$, we find that if we sweep it even a tiny amount further, then no solution along it lies inside the feasible region. This line is depicted by the solid red line (we have also shown here again the vector indicating the direction $\mathbf{c}$). The red point $\mathbf{x}^\mathsf{T} = (3, 5)$ is the *only* point in the feasible region lying on this final line, so

[1]Here, we have plotted only the integer values, but you should always imagine this happening *continuously*. It just so happens that here our optimal objective takes an integer value 13, but this will not always be the case.

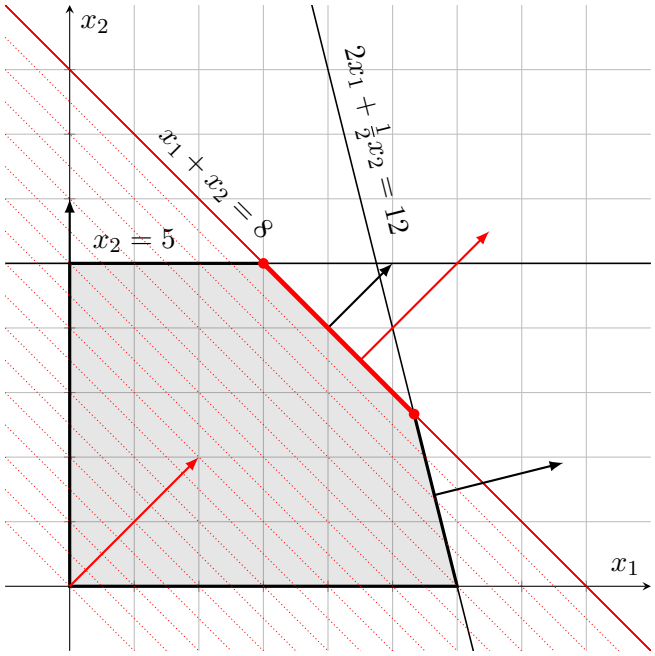
it is our *optimal* solution, and has value $\mathbf{c}^\mathsf{T}\mathbf{x} = 13$. Going back to the definition of an optimal solution, notice that indeed every solution $\mathbf{x}'$ in the feasible region of our program lies the side of the line through $\mathbf{x}$ opposite the direction $\mathbf{c}$, and so has $\mathbf{c}^\mathsf{T}\mathbf{x}' \leq z = \mathbf{c}^\mathsf{T}\mathbf{x}$.

Recall that we say a general inequality of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$ is *tight* if in fact $\mathbf{a}^\mathsf{T}\mathbf{x} = b$. Graphically, if some solution $\mathbf{x}$ makes an inequality tight, then it must lie on the boundary line for that constraint. In our picture, our feasible solution is a point $\mathbf{x}$ at which 2 constraints are tight—in particular $x_1 + x_2 = 8$ and $x_2 = 5$. In general, it seems reasonable to expect that our solutions will always lie at some *corner point* of the feasible region, and because we are in 2 dimensions, it will take 2 lines to uniquely identify any such point.

A potential issue may occur, though, if our vector $\mathbf{c}$ points in exactly the same direction as some constraint's vector $\mathbf{a}$. For example, suppose we change our objective function to be:

$$2x_1 + 2x_2$$

Then, $\mathbf{c}^\mathsf{T} = (1, 1)$ and our picture changes as follows (again, we draw lines for points with objective value $1, 2, \ldots$).



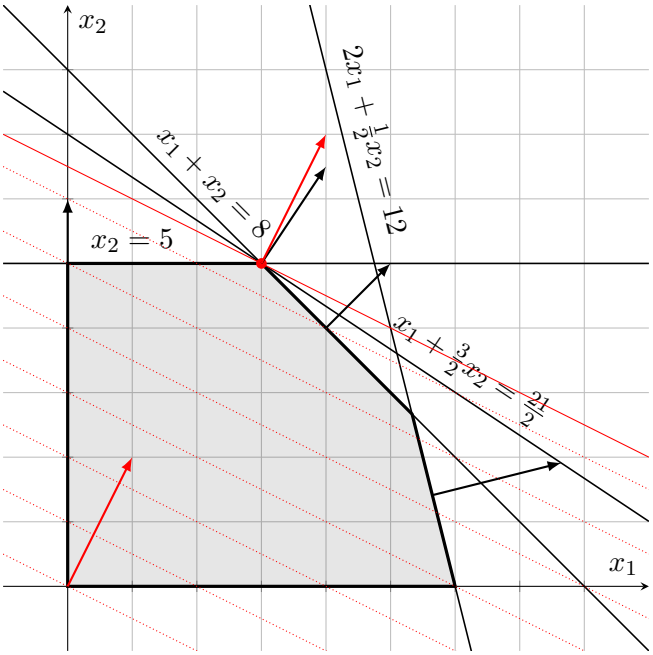
Because our objective was exactly perpendicular to the line of some constraint, *any point* along the relevant segment of this constraint will be optimal. However, each endpoint of this line will be a corner point, so we might as well return one of those. Thus, our intuition that we only need to worry about corner points ends

up being okay, as long as we keep in mind that, when situations like this arise, the corner point we look at might not be a *unique* optimal solution.

There are a few other subtleties that we now consider. First, notice that some constraint might be *redundant*, and not actually affect the feasible region at all! Suppose we add one more constraint to our linear program to obtain:

$$\begin{aligned} \text{maximize} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & x_1 + \frac{3}{2}x_2 \leq \frac{21}{2} \\ & x_2 \leq 5 \\ & x_1 + x_2 \leq 8 \\ & 2x_1 + \frac{1}{2}x_2 \leq 12 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Our sketch will now look like this:



Notice that the extra constraint didn't actually change the feasible region, because everything was already on the correct side of this line anyway. However, it happened that the line for this constraint went right through a corner of the feasible region. Thus, this point will now have *more than* 2 tight constraints. We know, however, that one of them isn't really needed. We'll see next week that this situation is called *degeneracy*.

Next, note that we need to be careful when we deal with *lower bound* constraints. Let's start again with program 3.1, but let's add a constraint of the form:

$$-\frac{1}{6}x_1 + x_2 \geq 0\,.$$

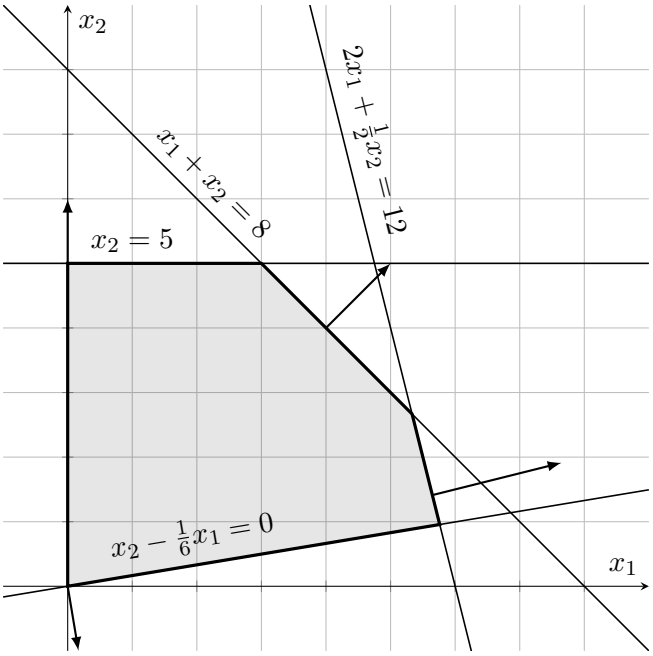
We get:

$$\begin{aligned} \text{maximize} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & -\frac{1}{6}x_1 + x_2 \geq 0 \\ & x_2 \leq 5 \\ & x_1 + x_2 \leq 8 \\ & 2x_1 + \frac{1}{2}x_2 \leq 12 \\ & x_1, x_2 \geq 0 \end{aligned}$$

We need to be careful that we shade the correct side of the line corresponding to this new constraint! The easiest way to avoid mistakes is to always convert your program to standard inequality form. This will turn all constraints into $\leq$ constraints. Then, if you draw the normal vector $\mathbf{a}$ from each plane, you should always shade the side *opposite* the direction that $\mathbf{a}$ is pointing. In this case, we rewrite our program, multiplying the first constraint by -1 to get:

$$\begin{aligned} \text{maximize} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & \frac{1}{6}x_1 - x_2 \leq 0 \\ & x_2 \leq 5 \\ & x_1 + x_2 \leq 8 \\ & 2x_1 + \frac{1}{2}x_2 \leq 12 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Now, notice that the first constraint has $\mathbf{a}^\mathsf{T} = (1/6, -1)$. When we sketch this region, we will get:



3.1 Infeasible and Unbounded Linear Programs

Now, we note that there are some cases in which an optimal solution may not exist. Intuitively, Definition 2.2 can fail for one of two reasons: (1) there are no feasible solutions to our program at all, or (2) for any feasible solution $\mathbf{x}$, we can always find a feasible solution $\mathbf{y}$ that is better. In these 2 cases, we call the linear program *infeasible* or *unbounded*, respectively.

Definition 3.1 (Infeasible)**.** We say that a linear program is *infeasible* if it has no feasible solutions.

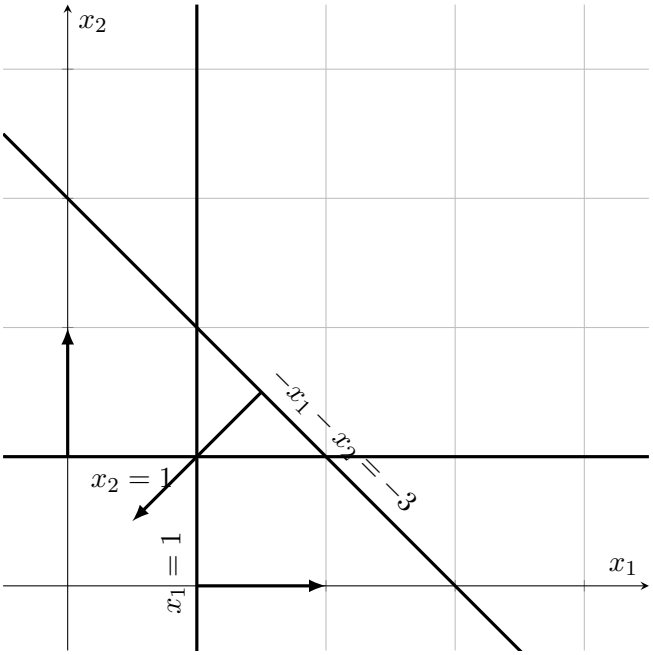
Definition 3.2 (Unbounded)**.** We say that a linear program in standard inequality form is *unbounded* if for *every* $k \in \mathbb{R}$, we can find a feasible solution $\mathbf{x}$ to the program for which $\mathbf{c}^\mathsf{T}\mathbf{x} \geq k$.

You can intuitively think of an unbounded linear program as having an "infinite" possible objective value. Indeed, Definition 3.2 says that no matter how big we choose k, there is always a feasible solution better.

Example 3.1. The following linear program is *infeasible*:

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{subject to} \quad & -x_1 - x_2 \leq -3 \\ & x_1 \leq 1 \\ & x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

When we try to sketch the constraints for this linear program we get something like this:

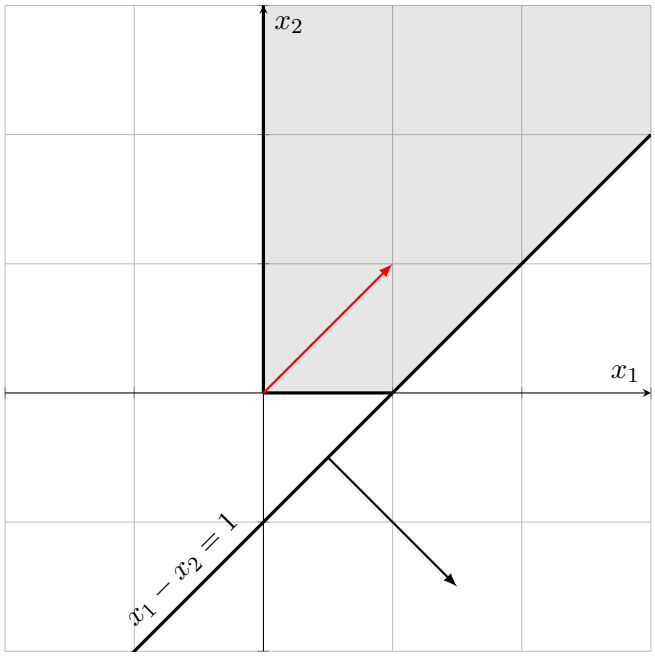


Each constraint is a $\leq$, so as before we want to shade the region *opposite* the normal vector of each line. Notice that there is no point that is on the correct side of all three constraints at once! So, the feasible region is empty, which means our program is *infeasible*.

Example 3.2. The following linear program is *unbounded*, since we can make the objective arbitrarily large: for any proposed "bound" $k \geq 0$ we can set $x_1 = 1$ and $x_2 = k$. Then, $x_1 - x_2 = 1 - k \leq 1$, so this solution is feasible, but it has objective value of $k + 1$.

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 - x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

We can see this easily by sketching the linear program and its objective:

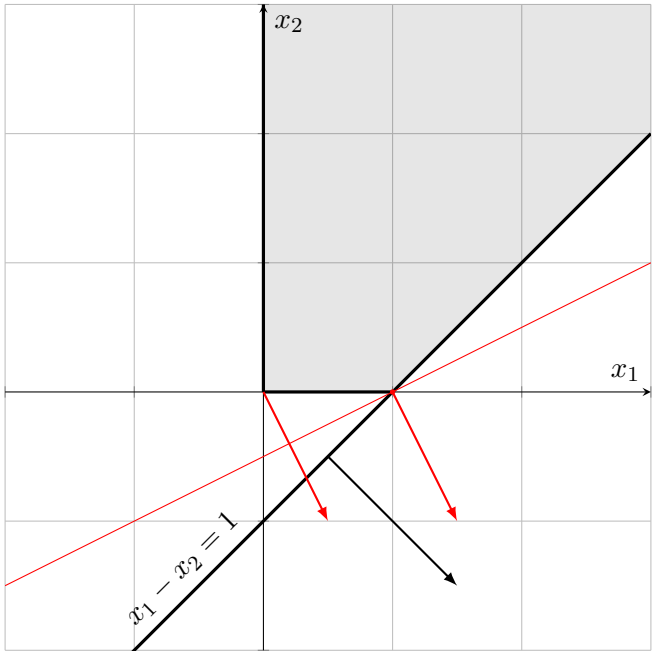


Clearly, if we keep moving in the direction of the objective, we will never stop. Notice that the issue of whether or not a linear program is *feasible* or *infeasible* is purely a question about its constraints. However, to decide if a program is *unbounded*, we also need to take its objective into account.

Example 3.3. Although it has exactly the same constraints as the previous program, the following linear program is neither unbounded nor infeasible.

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}x_1 - x_2 \\ \text{subject to} \quad & x_1 - x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Again, this becomes clear when we sketch the program. We find that the optimal solution is the point farthest in the direction of the red vector, which is the corner point $(1, 0)$. Here, we moved in a *downwards direction*, because that's where the objective pointed.



3.2 Three Dimensions

Everything we have just discussed carries over into higher dimensions as well. However, our intuition needs to be generalised slightly. An equation like $\mathbf{a}^\mathsf{T}\mathbf{x} = b$ still describes all the points $\mathbf{x}$ that we can reach by "walking along" in the direction $\mathbf{a}$ until we reach a distance of $\frac{b}{\|\mathbf{a}\|}$, and then walking any distance at right angles to $\mathbf{a}$. However, in three dimensions, we can move in several different directions

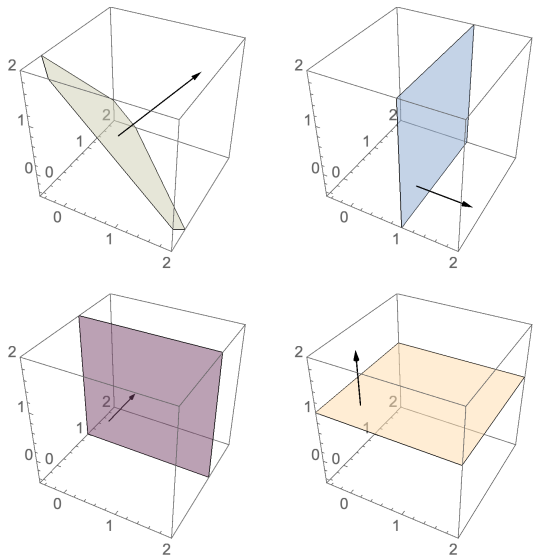


Figure 3.1: Constraints (3.2)–(3.5) in 3-dimensional space.

and still be at a right-angle to $\mathbf{a}$. Thus, the solutions to $\mathbf{a}^\mathsf{T}\mathbf{x} = b$ will be an entire 2-dimensional *plane* that intersects $\mathbf{a}$ at right angles. As before, we call the vector $\mathbf{a}$ the *normal vector* of this plane.

This means that now our systems of constraints are given by planes, and feasibly solutions are just those points that are on the correct side of all of them at once. For example, here is a set of linear constraints in 3 variables:

$$x_1 + x_2 + x_3 \leq 1.5 \tag{3.2}$$
$$x_1 \leq 1 \tag{3.3}$$
$$x_2 \leq 1 \tag{3.4}$$
$$x_3 \leq 1 \tag{3.5}$$

If we draw the boundary each constraint separately in space, together with its normal vector, we get the pictures shown in Figure 3.2. Notice that each constraint's boundary, given by the solutions to $\mathbf{a}^\mathsf{T}\mathbf{x} = b$, is a *plane* of all points $\mathbf{x}$ that are perpendicular (in some direction) to the vector $\mathbf{a}$.

As before, since our program is in standard form, our feasible region lies on the opposite side of each plane's normal vector. Here are 2 drawings of all 4 constraints together, and a drawing of the resulting feasible region (which is now a 3-dimensional region) from 2 different viewpoints:

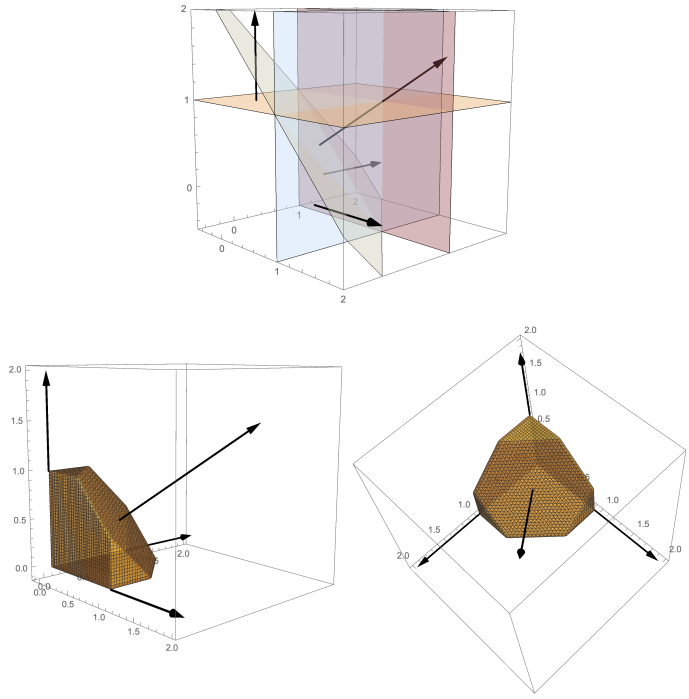


Figure 3.2: A views of all constraints (3.2)–(3.5) and two different views of the feasible region, together with each constraint's normal vector.

3.3 Convex Combinations and Extreme Point Solutions

We argued intuitively that in 2 and 3 dimensions, a feasible, bounded linear program would always have an optimal solution at a *corner point* of the feasible region. Now, our goal is to generalise these concepts to linear programs with more than 3 variables. Of course, a major difficulty is that it is difficult to think geometrically in more than 3 dimensions! In order to deal with this, we need to move past our intuition to more precise definitions.

We begin with the following notion, which defines a special kind of linear combination:

Definition 3.3 (Convex Combination)**.** We call a linear combination $a_1x_1 + a_2x_2 + \cdots + a_nx_n$ a *convex combination* of $x_1, x_2, \ldots, x_n$ if and only if:

- $a_i \geq 0$ for all $i = 1, 2, \ldots, n$.
- $a_1 + a_2 + \cdots + a_n = 1$.

Note that, as with linear combinations, our definition of convex combinations applies to vectors, as well. We can think of linear combinations of *weighted sums*. Then, convex combinations are just *weighted averages*, since their weights sum up to 1.

Typically, we will talk about convex combinations of a *pair of vectors*. In this case, our definition can be restated as follows:

Definition 3.4 (Convex Combination of 2 Vectors)**.** We say that $\mathbf{x}$ is a *convex combination* of two vectors $\mathbf{y}$ and $\mathbf{z}$ if and only if $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$ for some $\lambda \in [0, 1]$.

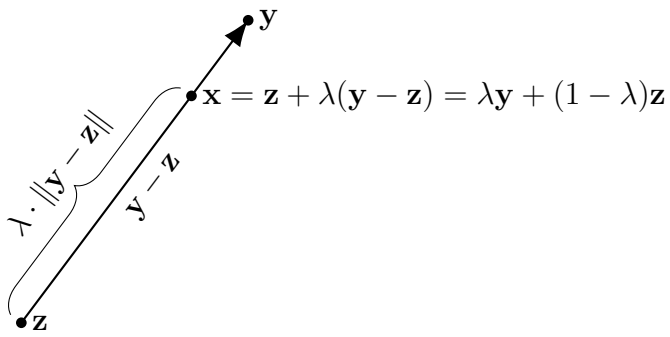
Notice that we only needed to give the coefficient λ of the first vector $\mathbf{y}$, since we knew that the second one had to be $(1-\lambda)$ in order to make our coefficients sum to 1 (as required by Definition 3.3).

Geometrically, if we view the vectors $\mathbf{y}$ and $\mathbf{z}$ as points in space, then a convex combination of the form $\lambda\mathbf{y} + (1-\lambda)\mathbf{z}$ is a point lying on the *line segment* joining $\mathbf{y}$ and $\mathbf{z}$. To see this, we can expand the equation $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$ to obtain

$$\mathbf{x} = \lambda\mathbf{y} + \mathbf{z} - \lambda\mathbf{z} = \mathbf{z} + \lambda(\mathbf{y} - \mathbf{z})\,.$$

Thus, the point $\mathbf{x}$ can be obtained by starting at the point $\mathbf{z}$ and then moving a λ fraction along the vector $(\mathbf{y} - \mathbf{z})$ that goes from $\mathbf{z}$ to $\mathbf{y}$. If $\lambda = 0$, we will stay

at the point $\mathbf{z}$ and if $\lambda = 1$, we will arrive at the point $\mathbf{y}$. For all other values of $\lambda \in (0, 1)$, we will arrive at some point on a line segment in between $\mathbf{y}$ and $\mathbf{z}$. This is shown in the following figure in which we have set $\lambda = 3/4$:



Definition 3.5. We say that a feasible solution $\mathbf{x}$ is an *extreme point solution* of a linear program if and only if it *cannot* be written as a convex combination $\lambda\mathbf{y} + (1 - \lambda)\mathbf{z}$ of two distinct feasible solutions $\mathbf{y}$ and $\mathbf{z}$ with $\mathbf{y} \neq \mathbf{x}$ and $\mathbf{z} \neq \mathbf{x}$ and $\lambda \in (0, 1)$.

This definition is about the *geometry* of our solution space: it says that extreme point solutions are precisely those that do not lie on any *line segment joining two other solutions*. This agrees with our intuition that optimal solutions of an LP will lie at the "corner points" of the feasible region, since we cannot draw a line segment from one point in the feasible region through a corner to some other point unless this other point is outside the feasible region. Thus, a corner point cannot be a convex combination of any pair of feasible solutions.

Week 4

Extreme Point and Basic Feasible Solutions

Last week, we investigated optimal solutions to linear programs by visualising the geometry of the problem, and we noticed that if a linear program has an optimal solution, then one of the "corners" of the feasible region is also an optimal solution. We formalised the idea of "corners" by defining extreme point solutions.

Unfortunately, it's not clear how we go about finding extreme points. First, we need a way to think *algebraically* about solutions to linear programs. In order to do this, we first show how to put any linear program into another useful form, which we call standard equation form.

Definition 4.1 (Standard Equation Form)**.** We say that a linear program is in *standard equation form* if it is written as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where A is an $m' \times n'$ matrix A of real numbers, $\mathbf{c} \in \mathbb{R}^{n'}$ and $\mathbf{b} \in \mathbb{R}^{m'}$.

We already know that any linear program can be written in standard inequality form, so it suffices to show how to transform such a linear program into equation form. Here, since we want to have equations, it makes sense to leave any equations in the original program as they are, rather than converting them to a pair of inequalities. If we carry out all of the other steps, making sure that the goal is maximise, the variables are all non-negative, and any inequalities are of the form

$\mathbf{a}^\mathsf{T}\mathbf{x} \leq \mathbf{b}$, we will get a program that looks like:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & P\mathbf{x} = \mathbf{d} \\ & \mathbf{x} \geq \mathbf{0} \ . \end{aligned} \tag{4.1}$$

where here $A\mathbf{x} \leq \mathbf{b}$ is a system of inequality constraints, and $P\mathbf{x} = \mathbf{d}$ is a system of equations, which remain since we did not convert them to inequalities. Consider any such linear program and let m be the number of inequality constraints (that is, the number of rows of A) and ℓ be the number of equations (that is, the number of rows of P). We need to convert our m inequalities to equations. The ith inequality constraint is given by:

$$A_{i,1}x_1 + A_{i,2}x_2 + \cdots + A_{i,n}x_n \leq b_i \, . \tag{4.2}$$

Let's introduce a new variable s_i to represent how much "slack" there is in this constraint. That is, s_i represents how far below b_i the right-hand side of (4.2) is:

$$s_i = b_i - (A_{i,1}x_1 + A_{i,2}x_2 + \cdots + A_{i,n}x_n) \tag{4.3}$$

Notice now that (4.2) is true if and only if $s_i \geq 0$. In other words, our ith inequality can be rewritten as an *equation*

$$A_{i,1}x_1 + A_{i,2}x_2 + \cdots + A_{i,n}x_n + s_i = b_i \tag{4.4}$$

together with a new non-negativity restriction $s_i \geq 0$. If we do this for each of our m constraints, introducing a new slack variable for each one, we get a new set of constraints that looks like:

$$\begin{array}{ccccccccccccccc} A_{1,1}x_1 & + & A_{1,2}x_2 & + & \cdots & + & A_{1,n}x_n & + & s_1 & & & & & = & b_1 \\ A_{2,1}x_1 & + & A_{2,2}x_2 & + & \cdots & + & A_{2,n}x_n & + & & s_2 & & & & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & & & \ddots & & & & \vdots \\ A_{m,1}x_1 & + & A_{m,2}x_2 & + & \cdots & + & A_{m,n}x_n & + & & & & & s_m & = & b_m \end{array} \tag{4.5}$$

So, we obtain a new linear program with $n + m$ variables. If we define:

$$Q = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} & 1 & 0 & \cdots & 0 \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} & 0 & 0 & \cdots & 1 \\ P_{1,1} & P_{1,2} & \cdots & P_{1,n} & 0 & 0 & \cdots & 0 \\ P_{2,1} & P_{2,2} & \cdots & P_{2,n} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ P_{\ell,1} & P_{\ell,2} & \cdots & P_{\ell,n} & 0 & 0 & \cdots & 0 \end{pmatrix}$$

$$\mathbf{x}' = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ s_1 \\ s_2 \\ \vdots \\ s_m \end{pmatrix} \qquad \mathbf{b}' = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \\ d_1 \\ d_2 \\ \vdots \\ d_\ell \end{pmatrix} \qquad \mathbf{c}' = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

then our constraints (4.5) can be written as $Q\mathbf{x}' = \mathbf{b}'$. Here, notice we obtained Q by placing an $m \times m$ identity matrix to the right of A, then placing an $\ell \times \ell$ all zero matrix to the right of P and then placing the first resulting matrix above the second. We can also write our problem's objective as $\mathbf{c}'^{\mathsf{T}}\mathbf{x}'$, which is simply equal to our original object $\mathbf{c}^{\mathsf{T}}\mathbf{x}$, since we added zeros to $\mathbf{c}$ for each of the new slack variables. Thus, we can formulate our program as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}'^{\mathsf{T}}\mathbf{x}' \\ \text{subject to} \quad & Q\mathbf{x}' = \mathbf{b}' \\ & \mathbf{x}' \geq \mathbf{0}\,, \end{aligned}$$

which is in the form required. Our new linear program has $n' = n + m$ variables (m more than the original program) and $m' = m + \ell$ constraints (the same number as the original program). Given a feasible solution $\mathbf{x}'^{\mathsf{T}} = (\mathbf{x}^{\mathsf{T}}; \mathbf{s}^{\mathsf{T}})$ of our new LP, consisting of values for both the original LP's decision variables $\mathbf{x}$ and our new slack variables $\mathbf{s}$, it is now easy to figure out which constraints of the original LP are made tight by $\mathbf{x}$: they are exactly those constraints whose corresponding slack variables are set to zero. From here on, when referring to a solution of a standard equation form LP, we will use the term "decision variable" to refer to those variables that were in the original LP, as opposed to "slack variables" which we have added to convert the LP into standard equation form.

The advantage of the standard equation form of a linear program is that it allows us to use methods for reasoning about and manipulating systems of linear equations, many of which you already know from linear algebra. (The advantage of the standard inequality form is that it usually has fewer variables for real-life problems and can be visualised more easily.) From now on, let's consider an arbitrary linear program that has been put into standard equation form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}\,. \end{aligned} \tag{4.6}$$

Let's suppose that this linear program has m constraints, and that none of these constraints can be written as a linear combination of the others. This last assumption means that the rank of A is m, and can be made without loss of generality: if any constraint is a linear combination of the others, then removing it does not change the set of solutions of the equations (and hence the set of feasible solutions of the linear program).

4.1 Optimal Solutions and Extreme Points

Our first major theorem shows that, indeed, it is sufficient to check only extreme point solutions if we want to find an optimal solution to a linear program. This confirms our intuition about only needing to check "corner points" in 2 dimensions.

Theorem 4.1. If a linear program (in standard equation form) has an optimal solution, then it has an optimal solution that is also an extreme point solution.

Proof. Consider a linear program in standard equation form and suppose that it has at least one optimal solution. We will show how to obtain an optimal solution that is also an extreme point solution. The main idea is as follows: if a solution $\mathbf{x}$ is *not* an extreme point solution, we claim that it is possible to construct a feasible solution $\mathbf{x}'$ from $\mathbf{x}$ such that $\mathbf{c}^\mathsf{T}\mathbf{x}' \geq \mathbf{c}^\mathsf{T}\mathbf{x}$ but $\mathbf{x}'$ has one additional entry x_i set to 0. Intuitively, if x_i is a slack variable, then this makes an extra constraint tight, and if x_i is a decision variables, then this means we made an extra restriction $x_i \geq 0$ tight. In either case, we get a feasible solution $\mathbf{x}'$ that is just as good as $\mathbf{x}$ but lies on the boundary of (at least) *one more* inequality than $\mathbf{x}$. Eventually this should give us enough tight constraints to uniquely define a single corner point, and then we will have an extreme point.

Formally, we begin by showing the claim:

Claim 4.2. Suppose that $\mathbf{x}$ is an optimal solution of some linear program in standard equation form. Then if $\mathbf{x}$ is not an extreme point, there must exist some $\mathbf{x}'$ such that: (1) $\mathbf{c}^\mathsf{T}\mathbf{x}' = \mathbf{c}^\mathsf{T}\mathbf{x}$, (2) $\mathbf{x}'$ is also feasible solution to the LP, and (3) in $\mathbf{x}'$, at least one more variable is set to 0 than in $\mathbf{x}$.

Proof. Since $\mathbf{x}$ is not an extreme point there must be some $\lambda \in (0, 1)$ and two feasible solutions $\mathbf{y}$ and $\mathbf{z}$ with $\mathbf{y} \neq \mathbf{z}$ so that $\mathbf{x} = \lambda\mathbf{y} + (1 - \lambda)\mathbf{z}$.

Since $\mathbf{x}$ is optimal, $\mathbf{c}^\mathsf{T}\mathbf{x} \geq \mathbf{c}^\mathsf{T}\mathbf{y}$ and $\mathbf{c}^\mathsf{T}\mathbf{x} \geq \mathbf{c}^\mathsf{T}\mathbf{z}$. If $\mathbf{c}^\mathsf{T}\mathbf{x} > \mathbf{c}^\mathsf{T}\mathbf{y}$ then:

$$\mathbf{c}^\mathsf{T}\mathbf{x} = \lambda\mathbf{c}^\mathsf{T}\mathbf{y} + (1 - \lambda)\mathbf{c}^\mathsf{T}\mathbf{z} < \lambda\mathbf{c}^\mathsf{T}\mathbf{x} + (1 - \lambda)\mathbf{c}^\mathsf{T}\mathbf{z} \leq \lambda\mathbf{c}^\mathsf{T}\mathbf{x} + (1 - \lambda)\mathbf{c}^\mathsf{T}\mathbf{x},$$

and so $\mathbf{c}^\mathsf{T}\mathbf{x} < \mathbf{c}^\mathsf{T}\mathbf{x}$; a contradiction. Thus $\mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{c}^\mathsf{T}\mathbf{y}$. Similarly, if $\mathbf{c}^\mathsf{T}\mathbf{x} > \mathbf{c}^\mathsf{T}\mathbf{z}$ then

$$\mathbf{c}^\mathsf{T}\mathbf{x} = \lambda\mathbf{c}^\mathsf{T}\mathbf{y} + (1-\lambda)\mathbf{c}^\mathsf{T}\mathbf{z} < \lambda\mathbf{c}^\mathsf{T}\mathbf{y} + (1-\lambda)\mathbf{c}^\mathsf{T}\mathbf{x} \leq \lambda\mathbf{c}^\mathsf{T}\mathbf{x} + (1-\lambda)\mathbf{c}^\mathsf{T}\mathbf{x},$$

and again we would have a contradiction $\mathbf{c}^\mathsf{T}\mathbf{x} < \mathbf{c}^\mathsf{T}\mathbf{x}$. Thus, $\mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{c}^\mathsf{T}\mathbf{z}$ as well. This shows that both $\mathbf{y}$ and $\mathbf{z}$ must be optimal solutions.

We now consider the line passing through $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$. We can represent any solution $\mathbf{x}'$ on this line as $\mathbf{x}' = \mathbf{x} + \theta(\mathbf{y} - \mathbf{z})$ for $\theta \in \mathbb{R}$. For any such solution:

$$\mathbf{c}^\mathsf{T}\mathbf{x}' = \mathbf{c}^\mathsf{T}(\mathbf{x} + \theta(\mathbf{y} - \mathbf{z})) = \mathbf{c}^\mathsf{T}\mathbf{x} + \theta\mathbf{c}^\mathsf{T}\mathbf{y} - \theta\mathbf{c}^\mathsf{T}\mathbf{z} = \mathbf{c}^\mathsf{T}\mathbf{x} + \theta\mathbf{c}^\mathsf{T}\mathbf{x} - \theta\mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{c}^\mathsf{T}\mathbf{x},$$

where the third equation follows from the fact that $\mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{c}^\mathsf{T}\mathbf{y} = \mathbf{c}^\mathsf{T}\mathbf{z}$, as we showed above. Thus, we know that every solution $\mathbf{x}' = \mathbf{x} + \theta(\mathbf{y} - \mathbf{z})$ on our line has the same objective value as $\mathbf{x}$ and so satisfies part (1) of the claim.

For part (2), we need to set θ so that $\mathbf{x}'$ satisfies $A\mathbf{x}' = \mathbf{b}$ and $\mathbf{x}' \geq \mathbf{0}$. of our linear program. First, we note that for *any* choice of θ:

$$A\mathbf{x}' = A(\mathbf{x} + \theta(\mathbf{y} - \mathbf{z})) = A\mathbf{x} + \theta A\mathbf{y} - \theta A\mathbf{z} = b + \theta b - \theta b = b,$$

where the third equation follows from the fact that $\mathbf{x}, \mathbf{y}$, and $\mathbf{z}$ are all feasible solutions. Now we just need to choose θ so that the $\mathbf{x}' \geq \mathbf{0}$ (so part (2) of the claim is true) and $\mathbf{x}'$ has at least one more 0 entry than $\mathbf{x}$ (so part (3) of the claim is true).

First, let's show that for any coordinate i with $x_i = 0$, we have $x'_i = 0$ for every possible choice of θ. Note that for any such x_i, since $\mathbf{x}$ is a convex combination of feasible solutions $\mathbf{y}$ and $\mathbf{z}$, we must have

$$x_i = \lambda y_i + (1-\lambda)z_i,$$

and $y_i \geq 0$ and $z_i \geq 0$. If either $y_i > 0$ or $z_i > 0$, we can see that we would have $x_i > 0$ as well. Thus, for any i with $x_i = 0$, we also have $y_i = 0$ and $z_i = 0$, and so:

$$x'_i = x_i + \theta(y_i - z_i) = 0 + \theta(0 - 0) = 0$$

for *any* choice of θ. This shows that changing θ cannot cause the number of coordinates set to 0 to decrease. We now need to show that we can choose θ so that one of the non-0 coordinates of $\mathbf{x}'$ becomes 0, but all the others stay non-negative. This will imply both that $\mathbf{x}' \geq \mathbf{0}$ (and so $\mathbf{x}$ is feasible) and part 3 of the claim.

Intuitively, we can just find a the point where this line passes through the boundary of an extra constraint. Let's consider how the coordinates of x'_i change as we vary θ. We have:

$$\frac{d}{d\theta}x'_i = \frac{d}{d\theta}\left(x_i + \theta(y_i - z_i)\right) = y_i - z_i.$$

Note that this is independent of θ, so every coordinate x_i will change at some constant rate as we change θ. Note that if $z_i > y_i$, then $\frac{d}{d\theta}x'_i = y_i - z_i < 0$ and so x'_i will decrease as we *increase* θ. Similarly, if $z_i < y_i$ then x_i will decrease as we *increase* θ. We can thus choose a direction to change θ (i.e. choose to either increase or decrease θ) so that at least 1 coordinate x'_i is decreasing. Then, we change θ in this direction until the *first moment* at which some coordinate x'_i becomes 0. For this value of θ, an extra coordinate (namely, x'_i) has become 0 and no other non-0 coordinate has yet become 0 (since we stopped changing θ at the *first* moment that this happened for *any* coordinate). This is exactly what was needed to complete the proof of parts 2 and 3 of the claim. □

The proof of Theorem 4.1 now follows inductively from the claim, as follows. Suppose that $\mathbf{x}$ is an optimal solution that is *not* an extreme point. Then, the claim implies that we can find a $\mathbf{x}'$ that is also feasible and optimal, but has 1 more variable set to 0. If $\mathbf{x}'$ is still not an extreme point, we can apply the claim again to it to set another variable to 0, continuing in this fashion until either we end up with some solution that is an extreme point, or *every* variable has been set to 0. In the latter case, we can also use the claim to show that we must in fact be at an extreme point. Indeed, suppose for the sake of contradiction that we were not at an extreme point but all variables had been set to 0; then, the claim says it must be able to set an *additional* variable to 0. But this is clearly not possible, since they are all 0 already. □

4.2 Basic Feasible Solutions

Theorem 4.1 suggests a way to reduce the *continuous* optimisation problem given by any linear program to a *discrete* optimisation problem—for linear programs that are not unbounded or infeasible, we can simply check all of the extreme points of a linear program and see which one gives the largest objective value. To do this, we need some method of *finding* extreme point solutions easily. The next definition will allow us to link the geometric notion of an extreme point to a precise algebraic notion. We will focus on solutions of 4.6 that have the following specific form.

Definition 4.2 (Basic Feasible Solution)**.** We say that $\mathbf{x}$ is a *basic feasible solution* of a linear program in standard equation form 4.6 if and only if $\mathbf{x}$ is feasible and the columns of A corresponding to the non-zero entries of $\mathbf{x}$ are linearly independent.

Note that since A has rank m, any set of more than m columns of A must be linearly dependent. Thus, a basic feasible solution can have at most m non-zero variables.

We will see that basic feasible solutions with fewer than m non-zero variables can cause problems. We call such basic feasible solutions *degenerate*:

Definition 4.3. We say that a basic feasible solution of a linear program in standard equation form is *non-degenerate* if it has *exactly* m non-zero variables, and *degenerate* if it has *fewer than* m non-zero variables (where, as usual, m is the number of constraints in the linear program).

We will call the m variables x_j corresponding to linearly independent columns of A *basic variables*, and the other variables *non-basic variables*. Note that non-basic variables will always be set to zero in the corresponding basic feasible solution.

The notion of a basic feasible solution will be useful to work with because it is *algebraic* rather than *geometric*. However, we can show that, in fact, it agrees with the geometric notion of an extreme point.

Theorem 4.3. A feasible solution $\mathbf{x}$ of a linear program in standard equation form 4.6 is a basic feasible solution if and only if it is an extreme point solution.

Proof. First, let's prove that every basic feasible solution $\mathbf{x}$ must be an extreme point solution. Let $\mathbf{x}$ be a basic feasible solution and let $B(\mathbf{x})$ be the set of columns of A corresponding to the non-zero entries of $\mathbf{x}$. Suppose, for the sake of contradiction, that $\mathbf{x}$ is *not* an extreme point; then, we can write $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$ for feasible solutions $\mathbf{y}$ and $\mathbf{z}$ with $\mathbf{y} \neq \mathbf{z}$ and some $\lambda \in (0, 1)$. Since $\mathbf{y}$ and $\mathbf{z}$ are both feasible solutions, we must have $A\mathbf{y} = \mathbf{b}$ and $A\mathbf{z} = \mathbf{b}$, and so:

$$A(\mathbf{z} - \mathbf{y}) = \mathbf{0}$$

The expression $A(\mathbf{z} - \mathbf{y})$ is a linear combination of some of the columns of A, specifically those corresponding to the non-zero entries of $\mathbf{z} - \mathbf{y}$. But, recall that $x_i = \lambda y_i + (1 - \lambda)z_i$, and $y_i \geq 0$, $z_i \geq 0$. Thus, as we noted in the proof of Theorem 4.1, the only way we can have $x_i = 0$ is if $y_i = 0$ and $z_i = 0$ as well. This in turn means that $z_i - y_i = 0$ and so the only columns of A that appear with *non-zero* coefficients in the linear combination $A(\mathbf{z} - \mathbf{y})$ must be those columns of $B(\mathbf{x})$. It follows that $A(\mathbf{z} - \mathbf{y})$ can be viewed as a linear combination of the columns in $B(\mathbf{x})$ and since $\mathbf{y} \neq \mathbf{z}$, we have $\mathbf{z} - \mathbf{y} \neq \mathbf{0}$, so this is a non-trivial linear combination. But this means we have found a non-trivial linear combination of the columns corresponding to $B(\mathbf{x})$ that is equal to zero. In other words, these columns are linearly dependent, contradicting our assumption that $\mathbf{x}$ was a basic feasible solution. Altogether, we have shown that if $\mathbf{x}$ is a basic feasible solution then $\mathbf{x}$ must be an extreme point solution.

For the other direction, suppose that $\mathbf{x}$ is *not* a basic feasible solution. We shall show that $\mathbf{x}$ is also *not* an extreme point solution. As in the previous case, let $B(\mathbf{x})$ be the set of columns of A corresponding to the non-zero entries of $\mathbf{x}$. Let's suppose these columns have indices $\{i_1, \ldots, i_r\}$. Then since $\mathbf{x}$ is not a basic feasible solution, the columns in $B(\mathbf{x})$ must be linearly dependent. That is, there must be some vector of constants $d_{i_1}, d_{i_2}, \ldots, d_{i_r}$ (one for each column of $B(\mathbf{x})$) not all equal to zero such that:

$$d_{i_1}\mathbf{a}_{i_1} + d_{i_2}\mathbf{a}_{i_2} + \cdots + d_{i_r}\mathbf{a}_{i_r} = \mathbf{0}\,,$$

We can extend the vector $\mathbf{d}$ to a vector in $\mathbb{R}^n$ by simply setting $d_i = 0$ for each index i corresponding to a column that is not in $B(\mathbf{x})$. Then, we have:

$$A\mathbf{d} = d_{i_1}\mathbf{a}_{i_1} + d_{i_2}\mathbf{a}_{i_2} + \cdots + d_{i_r}\mathbf{a}_{i_r} = \mathbf{0}\,.$$

It follows that for any constant $\theta \in \mathbb{R}$:

$$\begin{aligned} A\mathbf{x} + \theta A\mathbf{d} &= A\mathbf{x} + \mathbf{0} = A\mathbf{x} = \mathbf{b} \\ A\mathbf{x} - \theta A\mathbf{d} &= A\mathbf{x} - \mathbf{0} = A\mathbf{x} = \mathbf{b}, \end{aligned}$$

and so both $\mathbf{x} + \theta\mathbf{d}$ and $\mathbf{x} - \theta\mathbf{d}$ satisfy our problem's constraints. If we set $\theta > 0$ small enough, we can make sure that $\mathbf{x} + \theta\mathbf{d} \geq \mathbf{0}$ and $\mathbf{x} - \theta\mathbf{d} \geq \mathbf{0}$, as well. To see this, consider the non-zero entry d_i of $\mathbf{d}$ for which $\left|\frac{x_i}{d_i}\right|$ is the smallest (note that such an entry must exist, since there is at least one value $d_i \neq 0$). We set θ to this value $\left|\frac{x_i}{d_i}\right|$. Then, for any coordinate j with $d_j \neq 0$, we have $\theta \leq \left|\frac{x_j}{d_j}\right|$ and so $-x_j \leq \theta d_j \leq x_j$. It follows that for every coordinate j, either $d_j = 0$, in which case $x_j + \theta d_j = x_j + \theta \cdot 0 = x_j \geq 0$ and $x_j - \theta d_j = x_x - \theta \cdot 0 = x_j \geq 0$ (since $\mathbf{x}$ is feasible), or

$$\begin{aligned} x_j + \theta d_j &\geq x_j - x_j = 0 \\ x_j - \theta d_j &\geq x_j - x_j = 0\,. \end{aligned}$$

Thus, $\mathbf{x}+\theta\mathbf{d} \geq \mathbf{0}$ and $\mathbf{x}-\theta\mathbf{d} \geq \mathbf{0}$ and so both are feasible solutions to our program. Moreover, we can write $\mathbf{x}$ as a convex combination $\mathbf{x} = \frac{1}{2}(\mathbf{x} + \theta\mathbf{d}) + \frac{1}{2}(\mathbf{x} - \theta\mathbf{d})$. Thus, $\mathbf{x}$ is *not* an extreme point solution. Altogether, we have shown that if $\mathbf{x}$ is not a basic feasible solution then it must also not be an extreme point solution. □

In order to gain some intuition about Theorem 4.3, let's return to our 2-dimensional setting. There, we saw that, for a linear program in standard inequality form, we had $n = 2$ variables and the "corner points" of the feasible region were those where at least 2 of our constraints (of the form either $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$ or $x_i \geq 0$) were tight. When we convert such a linear program to standard equation

form, we will introduce m new slack variables (one for each constraint $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$). After doing this, we have that a constraint is tight if and only if some variable corresponding to the constraint is 0: for constraints of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$ in the original linear program to be tight, the corresponding slack variable in the equation form linear program must be 0, and for constraints of the form $x_j \geq 0$ in the original program to be tight, we must have $x_j = 0$. The standard equation form linear program has $n + m = 2 + m$ variables and if we have m linear constraints in our matrix A, at most m of the columns of A can be linearly independent. So, indeed any basic feasible solution must have at most m non-zero variables, and set the remaining $2 + m - m = 2$ variables to be zero. These 2 zero-valued variables then give us exactly our 2 tight constraints! Also, note that if our solution has *fewer than* m non-zero variables, it must be the case that *more than* 2 constraints are tight at the corresponding point in our feasible region. This gives us some intuition for what degenerate solutions look like.

In 2-dimensions, we saw that if our objective function was exactly perpendicular to some line defining a boundary of the feasible region, then we could have optimal solutions *anywhere along this line*. In that case, we noted that both the left and right corner point of the feasible region could be chosen, so we were still guaranteed to have a corner point that was optimal. The following corollary, which follows immediately from combining Theorems 4.1 and 4.3, extends this intuition into arbitrary finite dimensions, and allows us to restrict our attention to basic feasible solutions when searching for an optimal solution.

Corollary 4.4. If a linear program has an optimal feasible solution, then it has an optimal solution that is also a basic feasible solution.

4.3 Geometry and Basic Feasible Solutions

In the last few lectures, we developed geometric intuition that allowed us to find the optimal solution of linear programs with 2 and 3 variables. Then, we moved to higher dimensions by introducing definitions for extreme points. We saw that one of these extreme points will always be an optimal solution to a linear program, provided that the program is not infeasible or unbounded, and gave an algebraic characterisation of extreme points by introducing the notion of a basic feasible solution. At this point, we have reduced the problem of solving a linear program to the problem of finding the *best* basic feasible solution of that program. Before going further, let's see an example bringing together some of these principles.

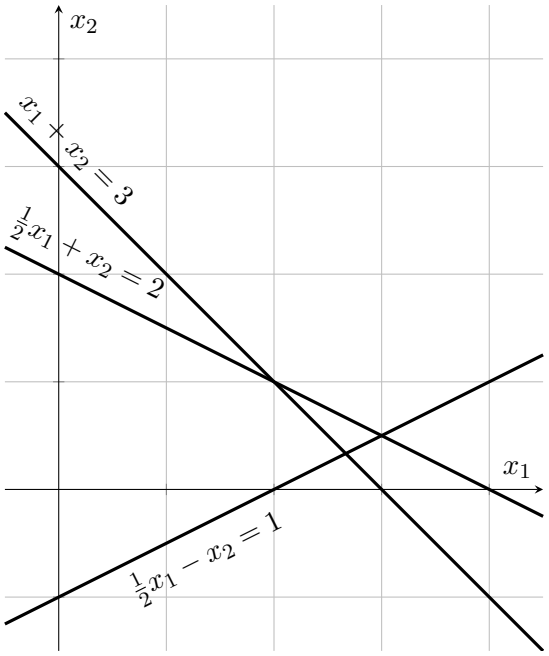
Let's consider the following linear program in 2 variables:

$$\begin{aligned} \text{maximize} \quad & 4x_1 + \frac{1}{2}x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 3 \\ & \frac{1}{2}x_1 + x_2 \leq 2 \\ & -\frac{1}{2}x_1 + x_2 \geq -1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Notice that the last constraint that is a $\geq$ constraint instead of a $\leq$. Before sketching, we should always convert all of our constraints to $\leq$ constraints, so we multiply both sides by -1. We get the following equivalent linear program in standard inequality form:

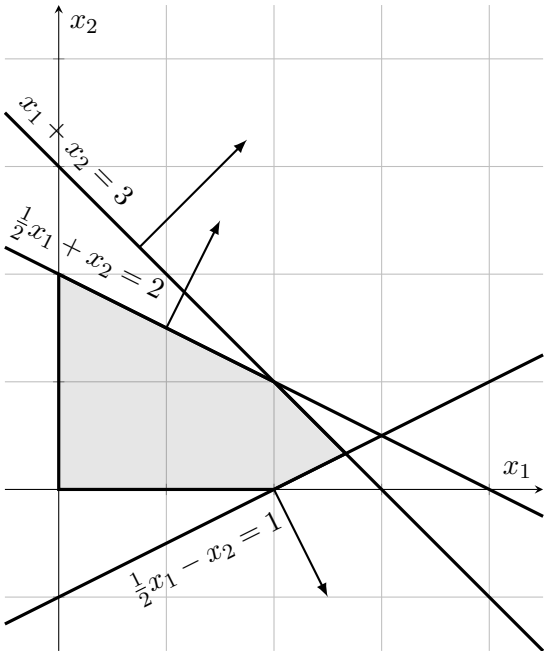
$$\begin{aligned} \text{maximize} \quad & 4x_1 + \frac{1}{2}x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 3 \\ & \frac{1}{2}x_1 + x_2 \leq 2 \\ & \frac{1}{2}x_1 - x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{4.7}$$

We want to see what the feasible region looks like, so we plot the boundary of each constraint. That is, we plot the lines we get when we change the $\leq$ signs into $=$ signs. We get the following:



Which part of the picture corresponds to the feasible region? We need to figure out which side of each line the feasible region lies on. One easy way is to pick any point we want on each line and then draw the normal vector for our line. For example, line $\frac{1}{2}x_1 + x_2 = 2$ is a linear equation of the form $\mathbf{a}^\mathsf{T}\mathbf{x} = 2$, where $\mathbf{a}^\mathsf{T} = (\frac{1}{2}, 1)$ and $\mathbf{x} = \big((x_1, x_2)\big)$. So, we pick any point on this line and draw a vector starting at the point and moving 1/2 unit to right (corresponding to the 1/2 in the first coordinate of $\mathbf{a}$) and 1 up (corresponding to the 1 in the second coordinate of $\mathbf{a}$). Intuitively, the vector we have just drawn indicates the direction in (x_1, x_2) plane that corresponds to making the expression $\mathbf{a}^\mathsf{T}\mathbf{x} = \frac{1}{2}x_1 + x_2$ *larger*. Our line shows us the values for which this expression is equal to 2. Because we want to also include values that are *smaller than* 2, our feasible solutions must all lie on the *opposite* side of our normal vector $\mathbf{a}$.

If we repeat the same process for each different line, and consider the intersection of all these regions, together with the non-negativity restrictions on x_1 and x_2, we get a picture that looks like this:



Normally, we would finish by sketching the direction $\mathbf{c}$ corresponding to the objective function (here, $\mathbf{c}^\mathsf{T} = (4, 1/2)$) and use it to solve our program. Instead, let's see how this might be done using *calculation* and our notion of basic feasible solutions. We saw in the last lecture that any linear program that has an optimal solution must have an optimal solution that is also a basic feasible solution. But, what do these basic feasible solutions look like? We saw that they correspond to extreme points, which are like corners of our feasible region. Let's see in more detail exactly how this correspondence works.

In order to talk about basic feasible solutions, we need to rewrite our program into standard equation form. We have 3 inequalities in 4.7. We introduce a new slack variable for each of them to get the following program:

$$\begin{aligned} \text{maximize} \quad & 4x_1 + \frac{1}{2}x_2 \\ \text{subject to} \quad & x_1 + x_2 + s_1 = 3 \\ & \frac{1}{2}x_1 + x_2 + s_2 = 2 \\ & \frac{1}{2}x_1 - x_2 + s_3 = 1 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0 \end{aligned} \tag{4.8}$$

We used to have $n = 2$ variables and $m = 3$ constraints. Now, we have $n + m = 5$ variables: 2 original variables plus one new slack variable corresponding to each constraint. We can write our system of equations more succinctly in matrix form

as:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 1 & 0 \\ \frac{1}{2} & -1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}.$$

Remember that in a basic feasible solution, we require that all the variables that take non-zero values correspond to a linearly independent set of columns in the matrix above. Since our matrix has $m = 3$ rows, at most $m = 3$ columns can be linearly independent, and we should expect that any basic feasible solution will select at most 3 variables to be set to non-zero values, or, equivalently, selecting 3 linearly independent columns of our matrix. Suppose that we select columns 1, 2, and 5 (note that these are indeed linearly independent). Then, in a corresponding basic feasible solution we are allowing x_1, x_2 and s_3 to take non-zero values, and requiring that s_1 and s_2 be set to zero. We say that x_1, x_2, and s_3 are the *basic* variables in this solution, and that s_1 and s_2 are the *non-basic* variables.

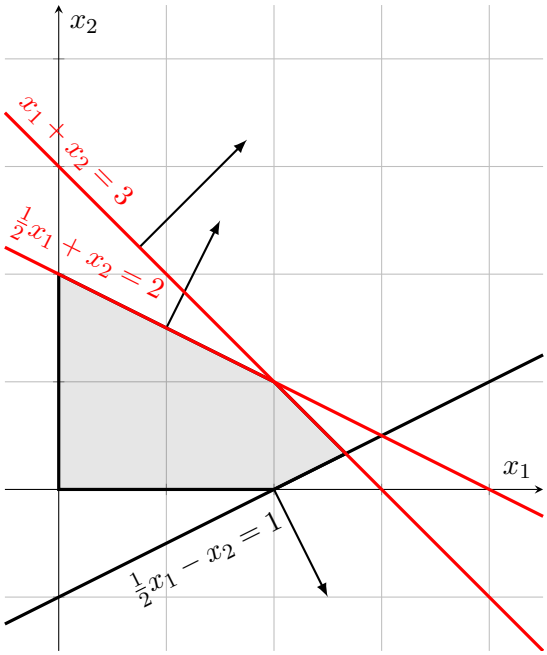
Let's now see what the effect of this choice is. Since we have $s_1 = 0$, then the first equation in 4.8 reads:

$$x_1 + x_2 = 3$$

This is exactly the same as saying that the constraint corresponding to slack variable s_1 is tight in our original program (4.7). Similarly, since we have $s_2 = 0$, then we must have

$$\frac{1}{2}x_1 + x_2 = 2$$

and so our second constraint in (4.7) is tight. Consider the following picture, in which we have drawn (in red) the equations corresponding to each of these tight constraints:



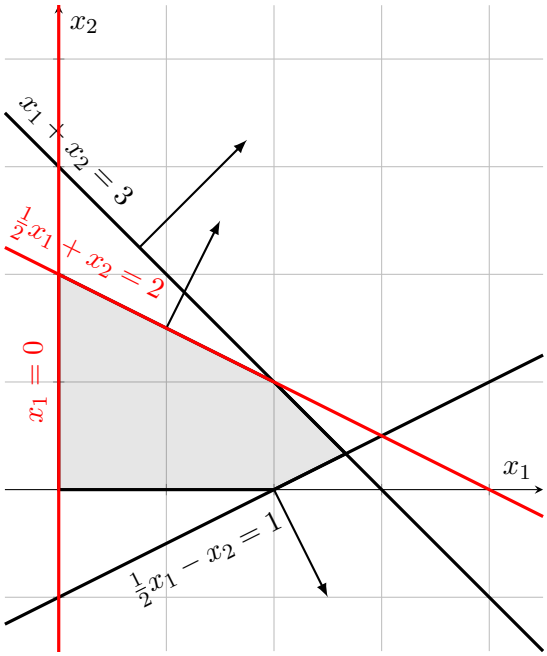
If we set s_1 and s_2 to zero and write out the system of constraints for our standard equation form program (4.8), we get:

$$\begin{aligned} x_1 + x_2 &= 3 \\ \frac{1}{2}x_1 + x_2 &= 2 \\ \frac{1}{2}x_1 - x_2 + s_3 &= 1 \end{aligned}$$

We find that this set of equations has a unique solution, namely $x_1 = 2$, $x_2 = 1$, $s_3 = 1$. This, then is our basic feasible solution, which was obtained by selecting a linearly independent set of columns 1,2, and 5 and then letting x_1, x_2, s_3 be our basic variables. If we plot just the x_1 and x_2 values for this solution on our picture, we get exactly the intersection point of these 2 red lines, which is also an extreme point of the feasible region. We showed in the last lecture that this will *always* happen—basic feasible solutions correspond exactly to extreme points of our feasible region.

So, we see that whenever a slack variable in our standard equation form program (4.8) is non-basic (that is, set to zero), it means that its corresponding constraint in our original program (4.7) is tight, and that this means that x_1 and x_2 must lie on this constraint's boundary line. What if one of x_1 or x_2 is set to zero? Remember that we actually have 2 extra boundaries in our feasible region, corresponding to the restrictions $x_1 \geq 0$ and $x_2 \geq 0$ in both of our programs. If, for example $x_1 = 0$, then one of *these* is tight. Indeed, suppose that we choose

columns 2,3, and 5 in our matrix to be basic. Then, in any corresponding basic feasible solution, we must have $x_1 = 0$ and $s_2 = 0$ (so $\frac{1}{2}x_1 + x_2 = 2$). If we draw these tight restrictions/constraints, we see:

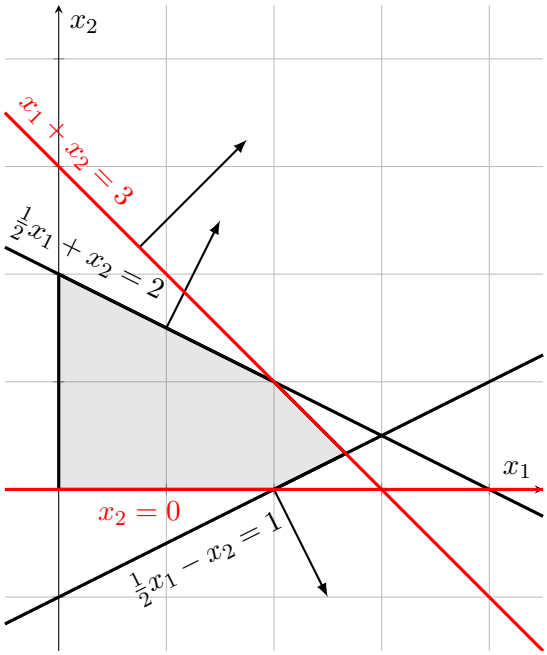


Again, we see that this gives us an extreme point of our feasible region. If we set x_1 and s_1 to zero our constraint equations become:

$$\begin{aligned} x_2 + s_1 &= 3 \\ x_2 &= 2 \\ -x_2 + s_3 &= 1 \end{aligned}$$

which has a unique solution $x_2 = 2$, $s_1 = 1$, $s_3 = 3$. Looking at the variables x_1 and x_2 from the original problem (4.7), we find $x_1 = 0$, $x_2 = 2$. So, again, our basic feasible solution lies at the intersection of 2 lines, which is an extreme point of the feasible region.

As a last example, let's suppose we choose columns $1, 4$ and 5 of our matrix. This will give us basic variables x_1, s_2, s_3 and non-basic variables x_2, s_1. As before, we will have 1 tight constraint, giving us $x_1 + x_2 = 3$, and one tight restriction, giving us $x_2 = 0$. When we draw these get:



Now, we find that our intersection point is strictly *outside* the feasible region. How could this be? Let's check our equations. If we set x_2 and s_1 to zero, we get:

$$\begin{aligned} x_1 &= 3 \\ \frac{1}{2}x_1 + s_2 &= 2 \\ \frac{1}{2}x_1 + s_3 &= 1 \end{aligned}$$

These equations have a unique solution $x_1 = 3$, $s_2 = \frac{1}{2}$, $s_3 = -\frac{1}{2}$. Indeed, if we check, the point $x_1 = 3$, $x_2 = 0$ corresponds to the intersection in our picture. However, notice that we can tell right away that this solution is not feasible, since it sets $s_3 = -\frac{1}{2}$, but (4.8) required that all variables were non-negative. The constraint in (4.7) that corresponds to the slack variable s_3 is $\frac{1}{2}x_1 - x_2 \leq 1$, and this is exactly the constraint that we are on the wrong side of!

Week 5

The Simplex Algorithm (I)

5.1 Intuition behind the Simplex Algorithm

Using the theorems from last week, we could attempt to solve a linear program as follows: try every possible way of choosing m basic variables, then solve the associated system of equations and check to make sure our solution is feasible. If we keep track of the best feasible solution we find (with respect to the objective $\mathbf{c}^\mathsf{T}\mathbf{x}$), we could figure out which one was optimal. Unfortunately, there are up to $\binom{n}{m} = \frac{n!}{(n-m)!m!}$ different such solutions, which grows extremely fast with n and m. Next we will see that it is possible to search for the best basic feasible solution in a more systematic way. This approach is called the *Simplex Algorithm*.

Eventually, we will see that there is a nice shorthand that can be used to implement the Simplex algorithm. Unfortunately, that shorthand gives little insight into *how* and *why* the algorithm works. Before introducing it, then, let's proceed in a more careful and "elementary" way, by looking at a specific example.

Consider our example linear program (4.8) from last week in standard equation form (note that here the distinction between slack variables and r we have just named the slacks as s_1, s_2, and s_3 instead of s_1, s_2, s_3):

$$\begin{aligned} \text{maximize} \quad & 4x_1 + \frac{1}{2}x_2 \\ \text{subject to} \quad & x_1 + x_2 + s_1 = 3 \\ & \frac{1}{2}x_1 + x_2 + s_2 = 2 \\ & \frac{1}{2}x_1 - x_2 + s_3 = 1 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0 \end{aligned}$$

The simplex algorithm works by moving from one basic feasible solution to another, so we will need to find *some* basic feasible solution to start off with. One

obvious set of linearly independent columns in our matrix is 3,4, and 5 (corresponding to the slack variables). If we set x_1 and x_2 to zero (corresponding to the origin of our picture from the previous lecture) and solve our equations for the remaining variables:

$$\begin{aligned} s_1 &= 3 \\ s_2 &= 2 \\ s_3 &= 1\,, \end{aligned}$$

which is feasible, since we have selected values that make the equations true, and all of these values were non-negative.

In general: if the right hand side of all of our equations is non-negative, we can always start by selecting our slack variables to be basic and find a feasible solution (we will see next week how to handle cases in which this doesn't work).

We obtained our values for s_1, s_2, and s_3 by solving equations, so instead of thinking about the solutions directly, we can instead think about the set of equations that lead us to them. From now on, let's use z to denote the value of our objective function, and write down an extra equation to remind us how to find the value of z. In our example, $z = 4x_1 + \frac{1}{2}x_2$. Then, let's rewrite our system of equations and our expression for z so that only basic variables (currently, $s_1, s_2.s_3$) are on the right-hand side and only constants and non-basic variables (currently, x_1, x_2) are on the left-hand side. If we use a line to separate the expression from z from the other equations, we get:

$$\begin{aligned} s_1 &= 3 - x_1 - x_2 \\ s_2 &= 2 - \frac{1}{2}x_1 - x_2 \\ s_3 &= 1 - \frac{1}{2}x_1 + x_2 \\ \hline z &= 4x_1 + \frac{1}{2}x_2 \end{aligned} \tag{5.1}$$

Then, since x_1 and x_2 are non-basic, we are thinking about a solution that sets them to zero. The we rewrote our equations then makes it easy to see that then $s_1 = 3$, $s_2 = 2$, $s_3 = 1$ and $z = 0$. Note, however, that we are *not* going to substitute zero for our non-basic variables in each step of our algorithm. Instead, we are going to keep a full, general set of equations relating all of the basic and non-basic variables. We will simply keep in mind that *if* we assign the non-basic variables (appearing on the right of our equations) a value of zero, then our current

set of equations will immediately tell us how the currently chosen basic variables (appearing on the left of the equations) should be set.

The key idea of the simplex algorithm is to perform a series of steps, each of which changes one variable from basic to non-basic and one variable from non-basic to basic. This exchange is done so that the objective function is always increasing (or, at least, never decreasing). Returning to our example, suppose we allowed one of x_1, x_2 to be basic, and increased its value. We see that if we increase x_1 or x_2, z will increase, since both of these variables has a positive coefficient in our expression for z. Let's pick x_1, since it has the largest coefficient and so will make z increase the fastest, and leave x_2 set to 0. In order to find a feasible solution, however, we need to make sure that: (1) our equations are satisfied (recall that these are just re-arranged versions of the original problem's equation constraints) and (2) no variable is assigned a negative value. The first equation in (5.1) reads:

$$s_1 = 3 - x_1 - x_2 \,.$$

In order to satisfy this equation as the value of x_1 increases, we must decrease the value of s_1 (recall that we are leaving $x_2 = 0$). Once the value of x_1 is larger than 3, we will have $s_1 < 0$. Thus, our first equation implies that if we keep $x_2 = 0$, we can increase the value of x_1 to at most 3 before our solution becomes infeasible.

Similarly, the next 2 equations say that the value of x_1 can be at most 4 (since if it were larger we would have $s_2 < 0$), and at most 2 (since if it were larger we would $s_3 < 0$). Altogether, we obtained 3 different bounds on how much we can increase the value of x_1. Of these, the strictest (that is, the *smallest*) bound is 2, corresponding to the last equation, so this is as far as we can increase x_1 while leaving x_2 set to zero. Since we want to increase z as much as possible, let's go ahead and set $x_1 = 2$. Notice that then the value of s_3 will become 0.

What we have done, then, is made one non-basic variable (x_1) take non-zero value, and one basic variable (s_3) take a zero value. We can think of this as constructing a new feasible solution in which x_1 has turned into a basic variable, and s_3 into a non-basic variable. In fact, this solution is also a basic feasible solution—you can check that the columns of our constraint matrix corresponding s_1, s_2 and x_2 are linearly independent.

Before, we had all of our basic variables on the left side of the equations and all of our non-basic variables on the right. Let's rearrange them again, so this is true for our new basic feasible solution. We start with the equation for the variable that has become non-basic (namely, s_3). Rearranging so that x_1 is on the right will give:

$$x_1 = 2 + 2x_2 - 2s_3 \,.$$

We can use this equation to rewrite the other 2, and the value for z, by simply

substituting its right hand side for x_1 everywhere. Our first equation becomes:

$$\begin{aligned} s_1 &= 3 - (2 + 2x_2 - 2s_3) - x_2 \\ &= 1 - 3x_2 + 2s_3 \end{aligned}$$

Our second equation becomes:

$$\begin{aligned} s_2 &= 2 - \frac{1}{2}(2 + 2x_2 - 2s_3) - x_2 \\ &= 1 - 2x_2 + s_3 \end{aligned}$$

Finally, our equation for z becomes:

$$\begin{aligned} z &= 4(2 + 2x_2 - 2s_3) + \frac{1}{2}x_2 \\ &= 8 + \frac{17}{2}x_2 - 8s_3 \end{aligned}$$

Altogether, we can now write our new system of equations as:

$$\begin{aligned} s_1 &= 1 - 3x_2 + 2s_3 \\ s_2 &= 1 - 2x_2 + s_3 \\ x_1 &= 2 + 2x_2 - 2s_3 \\ \hline z &= 8 + \frac{17}{2}x_2 - 8s_3 \end{aligned} \tag{5.2}$$

Notice that if we set the value of our non-basic variables (now x_2 and s_3) to zero, then $z = 8$ (as can easily be checked by the last equation). Thus, we have indeed found a better solution. Now, let's see if we can improve it again using the same idea. If we increase the value of either s_3 we will make z *smaller*, since it has a negative coefficient in our expression for z. However, we see that x_2 has a positive coefficient, so increasing the value of x_2 should make z larger.

As before, we keep the values of the other non-basic variables (here, s_3) set to zero, and see how much we can increase x_2 until the values for x_1, s_1, or s_2 become negative. The first equation doesn't give us *any* upper bound, since increasing x_2 only increases the right side. The second equation gives us an upper bound of $\frac{1}{3}$, and the last equation gives us an upper bound of $\frac{1}{2}$. Of these, $\frac{1}{3}$ is the strictest bound, corresponding to the second equation. So, we set the value of x_2 to $\frac{1}{3}$, turn s_1 into a non-basic variable. As before, we can rewrite the equation for s_1 get:

$$x_2 = \frac{1}{3} - \frac{1}{3}s_1 + \frac{2}{3}s_3\,.$$

Substituting the right side of this for x_2 in each of our other expressions as before, and simplifying we arrive at:

$$\begin{array}{rcccccc} x_2 &=& \frac{1}{3} &-& \frac{1}{3}s_1 &+& \frac{2}{3}s_3 \\ s_2 &=& \frac{1}{3} &+& \frac{2}{3}s_1 &-& \frac{1}{3}s_3 \\ x_1 &=& \frac{8}{3} &-& \frac{2}{3}s_1 &-& \frac{2}{3}s_3 \\ \hline z &=& \frac{65}{6} &-& \frac{17}{6}s_1 &-& \frac{7}{3}s_3 \end{array} \tag{5.3}$$

Notice that if we set the values of s_1 and s_3 zero, then we get $z = \frac{65}{6}$, which is again better than before. If we try to keep improving, we now see that increasing *any* of our non-basic variables x_2, s_3 will make z *worse*. It seems that we are stuck. Happily, this means that we have actually found the *optimal* solution for z.

Notice that we moved from (5.1) to (5.2) to (5.3) by simply rewriting equations. Thus, each of these systems of equations have exactly the same set of solutions. Moreover, the set of solutions to any of these systems of equations that also have $x_1, \ldots, s_3$ non-negative is exactly the same as the set of *feasible solutions* to (4.8). The equation for z in each case gives us a valid expression for the objective in terms of some of the variables—note we have more variables than equations, so it should make sense that there must be several ways to express this last line. In our last set of equations, we find that any solution of these equations with non-negative values for $x_1, \ldots, s_3$ must have $z \leq \frac{65}{6}$. In other words, any feasible solution of (4.8) has objective value at most $\frac{65}{6}$. It follows that our solution $\mathbf{x}^\mathsf{T} = (8/3, 1/3, 0, 1/3, 0)$ (which actually attains the objective value $\frac{65}{5}$) is an optimal solution to our problem.

Notice that at each step, we had a set of *basic variables* appearing on the left side of our equations, and a set of *non-basic variables* appearing on the right side of our equations. The operation that we performed in each step is called a *pivot*. As part of each pivot, we turned one non-basic variable into a basic variable. We call this the *entering variable* (since it enters the set of basic variables). We also turned one basic variable into a non-basic variable. This is called the *leaving variable* (since it leaves the set of basic variables).

5.2 The Simplex Algorithm in Tableau Form

The discussion in the previous section shows how tedious rewriting and keeping track of equations can be. It ends up that we can write all of the necessary

information for the simplex algorithm down in a concise form called a *tableau*. Suppose we want to solve a program in standard equation form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Our tableau will contain one row for each constraint, and an extra row for the objective function. Each row has a column for each variable, and then at the end we have a column for the the entries of A, the entries of $\mathbf{c}$ and the entries of $\mathbf{b}$ in one large table with general form:

A	$\mathbf{b}$
$\mathbf{c}$	$-z$

For example, the tableau for (4.8) looks like this:

1	1	1	0	0	3
$\frac{1}{2}$	1	0	1	0	2
$\frac{1}{2}$	-1	0	0	1	1
4	$\frac{1}{2}$	0	0	0	0

You have probably used something similar called an "augmented matrix" to perform Gaussian elimination. Each constraint of the form $a_1x_1 + \cdots + a_nx_n = b$ corresponds to one row of the tableau. In this row, we list only the values $a_1, \ldots, a_n$, and then the value for b. We use a vertical line to keep our constants b separate from the rest of the tableau. Additionally, we add an extra row corresponding to the objective function$c_1x_1 + c_2x_2 + \cdots + c_nx_n$. We list the coefficients $c_1, \ldots, c_n$ on this row, and separate it from the rest of the tableau by a line. The bottom right corner will always contain the value $-z$, (that is *minus* the current objective value). Each variable will correspond to a column of the tableau to the right of the vertical line. How do we find the basic variables in our tableau? They are the ones whose columns have exactly one entry that is 1 and all the rest 0's. That is, if we combine these columns in an appropriate order, we get an identity matrix. In our previous example, we chose our slack variables s_1, s_2, s_3 to be the first set of basic variables. Here, we have the same thing: notice that columns 3,4, and 5 can be combined to form an identity matrix. This means that x_1 and x_2 will be non-basic, and so we can see that our initial objective will have value $z = 0$ (remember, the slack variables do not contribute to the objective, which is why we put 0 in the bottom left corner. In order to help keep track of our basic variables,

let's introduce an extra column on the left. In this column, we will list the basic variable whose column has a 1 in that row. We also put a $-z$ in the last row, to remind us that the value on the right-hand side is the *negated* objective value of the current solution. Finally, let's label each main column in the tableau by the corresponding variable in the program. Here is what we get:

	x_1	x_2	s_1	s_2	s_3	
s_1	1	1	1	0	0	3
s_2	$\frac{1}{2}$	1	0	1	0	2
s_3	$\frac{1}{2}$	-1	0	0	1	1
$-z$	4	$\frac{1}{2}$	0	0	0	0

The algorithm now works as follows: we pick the entry with the largest positive coefficient in the last row of the tableau, and highlight its column, like so:

	x_1	x_2	s_1	s_2	s_3	
s_1	1	1	1	0	0	3
s_2	$\frac{1}{2}$	1	0	1	0	2
s_3	$\frac{1}{2}$	-1	0	0	1	1
$-z$	4	$\frac{1}{2}$	0	0	0	0

This tells us that x_1 will be our *entering variable* for this pivot operation. To find the leaving variable we go through the rows of the tableau (except for the last one below the line) one by one and examine the entry in the highlighted column. If this entry is positive, we divide the entry in the rightmost column (to the right of the vertical line) by it and record the answer. We get:

	x_1	x_2	s_1	s_2	s_3		
s_1	1	1	1	0	0	3	3
s_2	$\frac{1}{2}$	1	0	1	0	2	4
s_3	$\frac{1}{2}$	-1	0	0	1	1	2
$-z$	4	$\frac{1}{2}$	0	0	0	0	

Now, we highlight the row for which we recorded the *smallest* value, like so:

	x_1	x_2	s_1	s_2	s_3		
s_1	1	1	1	0	0	3	3
s_2	$\frac{1}{2}$	1	0	1	0	2	4
s_3	$\frac{1}{2}$	-1	0	0	1	1	2
$-z$	4	$\frac{1}{2}$	0	0	0	0	

This tells us that s_3 will be our *leaving variable*. We can now generate a new tableau. Since s_3 is leaving and x_1 is entering, we will replace s_3 by x_1 in our new tableau. Then, we perform a series of *elementary row operations* on our existing tableau in order to transform the blue column into a column with a 1 in the highlighted row and a zero everywhere else. Each such operation will either multiplies a row by a constant or adds a constant multiple of a row to another row.

If we multiply the third row by 2 we get:

	x_1	x_2	s_1	s_2	s_3	
s_1	1	1	1	0	0	3
s_2	$\frac{1}{2}$	1	0	1	0	2
x_1	1	-2	0	0	2	2
$-z$	4	$\frac{1}{2}$	0	0	0	0

Adding -1 times our new third row to the first row then gives:

	x_1	x_2	s_1	s_2	s_3	
s_1	0	3	1	0	-2	1
s_2	$\frac{1}{2}$	1	0	1	0	2
x_1	1	-2	0	0	2	2
$-z$	4	$\frac{1}{2}$	0	0	0	0

Adding $-\frac{1}{2}$ times our new third row to the second row then gives:

	x_1	x_2	s_1	s_2	s_3	
s_1	0	3	1	0	-2	1
s_2	0	2	0	1	-1	1
x_1	1	-2	0	0	2	2
$-z$	4	$\frac{1}{2}$	0	0	0	0

Finally, adding -4 times our new third row to the last row (corresponding to the objective), gives:

	x_1	x_2	s_1	s_2	s_3	
s_1	0	3	1	0	-2	1
s_2	0	2	0	1	-1	1
x_1	1	-2	0	0	2	2
$-z$	0	$\frac{17}{2}$	0	0	-8	-8

This is our next tableau. We can think of each row i in the middle part as an equation:

$$a_{i,1}x_1 + a_{i,2}x_2 + a_{i,3}s_1 + a_{i,4}s_2 + a_{i,5}s_3 = b_i\,,$$

where coefficients $a_{i,1}, \ldots, a_{i,5}$ are listed in the middle part, and the constant b_i is listed in right part. For example, the first row says that $3x_2 + s_1 - 2s_3 = 1$. Notice that this is equivalent to the equation we got using the previous method! You can check that in fact, the 3 equations corresponding to our tableau are exactly the same as the second set of equations we got when using the equation form of the simplex algorithm. The last row corresponds to an equation that says:

$$-z + c_1x_1 + c_2x_2 + \cdots + c_nx_n = d,$$

where z represents our objective value, each c_j is the entry in the jth column of this row, and d is the entry in the lower right of the tableau. Notice that when we rearrange, we get:

$$z = c_1x_1 + c_2x_2 + \cdots + c_nx_n - d,$$

which is why the entry on the last row corresponds to the *negated* value of the objective.

Now, let's continue. The only positive row coefficient in the last row is in column 2, so x_2 will be our entering variable. For each row above the line, we examine the entry in the highlighted column and, if this entry is positive, we divide the entry in the right-most column by it. Then, we highlight the row for which this gives the smallest value. We get:

	x_1	x_2	s_1	s_2	s_3		
s_1	0	3	1	0	-2	1	$\frac{1}{3}$
s_2	0	2	0	1	-1	1	$\frac{1}{2}$
x_1	1	-2	0	0	2	2	
$-z$	0	$\frac{17}{2}$	0	0	-8	-8	

So, x_2 is our entering variable and s_1 is our leaving variable. We now use elementary row operations to transform the highlighted column so that it has a 1 in the highlighted row, and a 0 everywhere else. We get:

	x_1	x_2	s_1	s_2	s_3	
x_2	0	1	$\frac{1}{3}$	0	$-\frac{2}{3}$	$\frac{1}{3}$
s_2	0	0	$-\frac{2}{3}$	1	$\frac{1}{3}$	$\frac{1}{3}$
x_1	1	0	$\frac{2}{3}$	0	$\frac{2}{3}$	$\frac{8}{3}$
$-z$	0	0	$-\frac{17}{6}$	0	$-\frac{7}{3}$	$-\frac{65}{6}$

At this step, there is no positive entries in the bottom part of our tableau. This means we have found an optimal solution. The solution sets each variable on the left-most side of the tableau to the value on the right-most side of the tableau, and sets all other variables to 0. Thus, as before, our solution is $\mathbf{x}^\mathsf{T} = (8/3, 1/3, 0, 1/3, 0)$. We can ignore our slack variables to get the optimal solution to the original linear program (4.7). This is the solution $x_1 = 8/3$, $x_2 = 1/3$. The objective value of this solution is given by -1 times the lower-right corner of the tableau. So, our optimal solution has objective value $65/6$.

Hopefully the simplex algorithm is clear from the above example. However, here is a long and precise general description of it (just in case):

Algorithm 5.1 (Simplex Algorithm for Problems with a Feasible Origin)**.** Suppose we are given a linear program in *standard inequality form*, with n variables and m constraints:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

This simplified version will only handle problems for which $\mathbf{b} \geq \mathbf{0}$ (we will see how to handle general problems later).

1. Translate the program to standard equation form by introducing a new, non-negative slack variable for each constraint. This will have the effect of introducing m new variables and appending an $m \times m$ identity matrix onto the right of A. Call the resulting $m \times (m+n)$ matrix A'.
2. Form the initial tableau from A', $\mathbf{c}$, and $\mathbf{b}$.
3. Repeatedly perform *pivot* operations, until the last row of the tableau has no positive entry.
4. When the last row of the tableau has no positive entry, stop. You have found an optimal solution. It sets each recorded variable on the far left to the value on the far right, and all other variables to 0. The objective value for this solution is given by -1 times the value in the bottom right corner of the tableau.

All that remains is to discuss how to form the initial tableau and how to carry out each pivot operation. We now give a general description of how to do both of these tasks.

Forming an Initial Tableau

1. Write down the matrix A' as the centre of the tableau. Draw lines to the left, right, and below this matrix.
2. To the right, write the column vector $\mathbf{b}$. We call this the *right* part of the tableau.
3. Below, write the vector $\mathbf{c}$. Add m zeros to the end of this vector (one for each slack variable). We call this the *bottom* part of the tableau.
4. To the left, write down the initial basic variables. They should be the slack variables, listed from top to bottom as $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$. We call this

the *left* part of the tableau.

5. Place a 0 in the bottom left corner, since the initial solution has objective value 0.

Performing a Pivot Operation Suppose that we have a current tableau with at least one positive entry in the bottom row. Then, the simplex algorithm requires that we carry out a *pivot* operation to produce a new tableau. We do this as follows:

1. Find the largest positive entry in the bottom part of the tableau (if there is a tie, choose the one furthest to the left). Circle or highlight the column of the tableau containing this entry. The variable corresponding to this row is the *entering* variable.
2. If *all* entries in the highlighted column are negative or zero, then stop. The linear program is unbounded. Otherwise, for each positive entry a in the highlighted column, divide each the entry in the right part of the tableau by a and record this value to the side.
3. Find the row for which you recorded the smallest entry (if there is a tie, choose the one closest to the top). Circle or highlight this row. The variable name in the left part of the tableau for this row is the *leaving* variable.
4. Form a new tableau. First, copy the names of the basic variables from the left part of your existing tableau to the left part of the new tableau, but write down the name of the entering variable in place of the name of the entering variable.
5. Multiply all entries the highlighted row of your existing tableau by a suitable value so that the element in both the highlighted row and highlighted column becomes 1. Copy the resulting row into the new tableau. Let's call this new row T.
6. For each other row R in your *existing* tableau, add to R a suitable multiple of T so that the element in the highlighted column of R 0. Copy the resulting row into your new tableau. **Remember:** you should include the value for the right part of the tableau in all of your row operations and you should also carry out this step for the row in the bottom part of the tableau!

Important: You should only be adding some multiple of *the pivot row* to each other row. In particular, you should never combine two non-pivot rows! This is a perfectly fine thing to do for Gaussian elimination but *not* for the simplex algorithm.

Example 5.1. Suppose we want to find an optimal solution of the following linear program:

$$\begin{aligned} \text{maximize} \quad & 2x_1 - x_2 + 8x_3 \\ \text{subject to} \quad & 2x_3 \leq 1 \\ & 2x_1 - 4x_2 + 6x_3 \leq 3 \\ & -x_1 + 3x_2 + 4x_3 \leq 2 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Solution. We first reformulate our problem in standard inequality form, by introducing a slack variable for each constraint. This gives us the following linear program:

$$\begin{aligned} \text{maximize} \quad & 2x_1 - x_2 + 8x_3 \\ \text{subject to} \quad & 2x_3 + s_1 = 1 \\ & 2x_1 - 4x_2 + 6x_3 + s_2 = 3 \\ & -x_1 + 3x_2 + 4x_3 + s_3 = 2 \\ & x_1, x_2, x_3, s_1, s_2, s_3 \geq 0 \end{aligned}$$

We can write our constraints in matrix form as:

$$\begin{pmatrix} 0 & 0 & 2 & 1 & 0 & 0 \\ 2 & -4 & 6 & 0 & 1 & 0 \\ -1 & 3 & 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

All the entries in our vector $\mathbf{b}$ (on the right of the $=$) are non-negative. Thus, we can take the slack variables s_1, s_2, s_3 corresponding to the last 3 columns of A as our initial basic solution. We form our simplex tableau as shown on the right, and compute the entering and leaving variables as shown on the left:

	x_1	x_2	x_3	s_1	s_2	s_3	
s_1	0	0	2	1	0	0	1
s_2	2	−4	6	0	1	0	3
s_3	−1	3	4	0	0	1	2
	2	−1	8	0	0	0	0

	x_1	x_2	x_3	s_1	s_2	s_3		
s_1	0	0	2	1	0	0	1	$\frac{1}{2}$
s_2	2	−4	6	0	1	0	3	$\frac{1}{2}$
s_3	−1	3	4	0	0	1	2	$\frac{1}{2}$
$-z$	2	−1	8	0	0	0	0	

The remaining pivot steps are as follows (again, we show how the entering and leaving variables are found on the right-hand side):

	x_1	x_2	x_3	s_1	s_2	s_3	
x_3	0	0	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$
s_2	2	-4	0	-3	1	0	0
s_3	-1	3	0	-2	0	1	0
	2	-1	0	-4	0	0	-4

	x_1	x_2	x_3	s_1	s_2	s_3		
x_3	0	0	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$	
s_2	2	-4	0	-3	1	0	0	0
s_3	-1	3	0	-2	0	1	0	
$-z$	2	-1	0	-4	0	0	-4	

	x_1	x_2	x_3	s_1	s_2	s_3	
x_3	0	0	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$
x_1	1	-2	0	$-\frac{3}{2}$	$\frac{1}{2}$	0	0
s_3	0	1	0	$-\frac{7}{2}$	$\frac{1}{2}$	1	0
$-z$	0	3	0	-1	-1	0	-4

	x_1	x_2	x_3	s_1	s_2	s_3		
x_3	0	0	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$	
x_1	1	-2	0	$-\frac{3}{2}$	$\frac{1}{2}$	0	0	
s_3	0	1	0	$-\frac{7}{2}$	$\frac{1}{2}$	1	0	0
$-z$	0	3	0	-1	-1	0	-4	

	x_1	x_2	x_3	s_1	s_2	s_3	
x_3	0	0	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$
x_1	1	0	0	$-\frac{17}{2}$	$\frac{3}{2}$	0	0
x_2	0	1	0	$-\frac{7}{2}$	$\frac{1}{2}$	1	0
$-z$	0	0	0	$\frac{19}{2}$	$-\frac{5}{2}$	-3	-4

	x_1	x_2	x_3	s_1	s_2	s_3		
x_3	0	0	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$	1
x_1	1	0	0	$-\frac{17}{2}$	$\frac{3}{2}$	2	0	
x_2	0	1	0	$-\frac{7}{2}$	$\frac{1}{2}$	1	0	
$-z$	0	0	0	$\frac{19}{2}$	$-\frac{5}{2}$	-3	-4	

	x_1	x_2	x_3	s_1	s_2	s_3	
s_1	0	0	2	1	0	0	1
x_1	1	0	17	0	$\frac{3}{2}$	2	$\frac{17}{2}$
x_2	0	1	7	0	$\frac{1}{2}$	1	$\frac{7}{2}$
$-z$	0	0	-19	0	$-\frac{5}{2}$	-3	$-\frac{27}{2}$

So, our optimal solution is $\mathbf{x}^\mathsf{T} = (17/2, 7/2, 0, 1, 0, 0)$. In terms of our original linear program, our solution sets $x_1 = 17/2$, $x_2 = 7/2$ and $x_3 = 0$. This solution has objective value $27/2$. □

Notice that in the last problem, we had several pivot steps in which we recorded a 0 to the right when computing the leaving variable. When we finished pivoting in these steps, the objective did not increase! These are called *degenerate* steps in the simplex algorithm, and we will talk more about them later.

5.3 Unbounded Linear Programs

Now, let's talk about some things that might go wrong in our algorithm. We have seen that if $c_j \leq 0$ for every entry in the bottom row, then there is no choice for our entering variable and this means our solution must be *optimal*. What if there is no way to choose our leaving variable?

Recall that once we have decided that our entering variable is x_e, we choose the leaving variable as follows: for each row i above the line in our tableau, if $a_{i,e} > 0$, we compute a bound $b_i/a_{i,e}$ and write it to the right of the tableau. Then, we choose a row with the smallest bound. The leaving variable is the variable that we have labelled that row with on the left. If $a_{i,e} \leq 0$ for every row i, we get no bounds at all and cannot choose a leaving variable! In this case, it ends up that the linear program is *unbounded*. Let us prove that this must be the case.

Suppose that the simplex algorithm stops with no way to choose a leaving variable it some step. We will show that in this case we can obtain a feasible solution $\mathbf{x}'$ with $\mathbf{c}^\mathsf{T}\mathbf{x}'$ at least k larger than our current feasible solution, for any value k.

Since we chose x_e is our entering variable we must have had $c_e > 0$. Consider *any* value $k > 0$, and set $x'_e = k/c_e$. Note that we will have $x'_e > 0$ since $c_e > 0$. Then, as usual, set $x'_j = 0$ for every other non-basic variable $j \neq e$. We set the remaining basic variables according to the equations in our tableau. The ith equation in our tableau will now be given by:

$$x'_i + a_{i,e}\frac{k}{c_e} = b_i\,, \tag{5.4}$$

since for every non-basic variable x_j we still have $x'_j = 0$, with the exception of x_e, which now has $x'_e = k/c_e$. Let's set the value of each basic variable x_i so that its corresponding equation is satisfied. Then, to show that we get a feasible solution, we need only argue that $x'_i \geq 0$ for every basic variable x'_i (note that our non-basic variables are all set to zero by construction). As we saw before, the simplex algorithm makes sure that always $b_i \geq 0$. We have just seen that $x_e \geq 0$ and since we couldn't compute a bound we must have had $a_{i,e} \leq 0$. Thus, $a_{i,e}x_e \leq 0$ and so we will set:

$$x'_i = b_i - a_{i,e}x_e \geq b_i = x_i\,. \tag{5.5}$$

Here, the first equation came from rearranging (5.4), the inequality from the fact that $a_{i,e}x_e \leq 0$ (as discussed above), and the final equation from the fact that $x_i = b_i$ in the solution represented by our initial tableau. Then, since $\mathbf{x}$ was feasible, we must have $x'_i \geq x_i \geq 0$ for every basic variable in our initial tableau. Thus, the solution is indeed feasible.

Let us now consider the objective value z' corresponding to our new solution. In the original simplex tableau, the objective function z satisfies the equation:

$$z = c_1 x_1 + \cdots + c_n x_n - d \tag{5.6}$$

where d is the entry in the bottom-right corner. In the tableau for $\mathbf{x}$, we had $c_i = 0$ for all basic variables x_i in this row, and for all non-basic variables x_j, we have $x_j = 0$. Thus, the equation (5.6) simply reads: $z = -d$, and so the objective value for $\mathbf{x}$ is $-d$. All of this remains true for $\mathbf{x}'$, except that we have set $x'_e = k/c_e$. Thus, the equation for z' will read:

$$z' = c_e \frac{k}{c_e} - d$$

Note that $c_e \frac{k}{c_e} = k$, and so we now have $z' = -d + k = z + k$. In summary, we have shown how to construct a feasible solution with objective value k larger than our current tableau's basic feasible solution *for any number* k. It follows that our linear program must be unbounded.

Week 6

The Simplex Algorithm (II)

Last week, we introduced the simplex algorithm, which gave us a general tool for solving linear programs that is more systematic than sketching (and scales easily to more than 3 variables). There, we focused on presenting the main ideas of the algorithm, first in terms of rewriting equations, and later in a concise tableau format. This week, we first review the algorithm from a geometric perspective.

6.1 Geometry of the Simplex Algorithm

First, let's return to our running example, which is the linear program:

$$\begin{aligned} \text{maximize} \quad & 4x_1 + \frac{1}{2}x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 3 \\ & \frac{1}{2}x_1 + x_2 \leq 2 \\ & \frac{1}{2}x_1 - x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

We saw that after adding slack variables and carrying out the simplex algorithm, we obtained the following sequence of tableaux:

	x_1	x_2	s_1	s_2	s_3	
s_1	1	1	1	0	0	3
s_2	$\frac{1}{2}$	1	0	1	0	2
s_3	$\frac{1}{2}$	-1	0	0	1	1
$-z$	4	$\frac{1}{2}$	0	0	0	0

	x_1	x_2	s_1	s_2	s_3	
s_1	0	3	1	0	-2	1
s_2	0	2	0	1	-1	1
x_1	1	-2	0	0	2	2
$-z$	0	$\frac{17}{2}$	0	0	-8	-8

	x_1	x_2	s_1	s_2	s_3	
x_2	0	1	$\frac{1}{3}$	0	$-\frac{2}{3}$	$\frac{1}{3}$
s_2	0	0	$-\frac{2}{3}$	1	$\frac{1}{3}$	$\frac{1}{3}$
x_1	1	0	$\frac{2}{3}$	0	$\frac{2}{3}$	$\frac{8}{3}$
$-z$	0	0	$-\frac{17}{6}$	0	$-\frac{7}{3}$	$-\frac{65}{6}$

Notice that the way we do our pivot operations ensures that for each basic variable, the corresponding column of the tableau will have a 0 in every row except for one, which has entry 1. This row is precisely the one we have labelled with that basic variable. Thus, each row of our tableau represents an equation relating a single basic variable to the non-basic variables. If B is the set of all indices of basic variables (so that $i \in B$ if and only if x_i is basic) then for each $i \in B$, we will get a single row of our tableau, given an equation of the general form:

$$x_i + \sum_{j \notin B} a_{i,j} x_j = b_i \,, \tag{6.1}$$

where $a_{i,j}$ is the entry in the jth column of the row labelled by x_i and b_i is the entry on the right side of the tableau for this row. Also, the last line gives us an expression for our objective function, formulated in terms of *only the non-basic variables*:

$$z = -d + \sum_{j \notin B} c_j x_j \,, \tag{6.2}$$

where c_j is the entry in the jth column of the bottom part of the tableau and d is the value in the lower-right corner.

Just like before, when all non-basic variables are set to zero, these equations say that $x_i = b_i$ for each $i \in B$ and $z = d$. Each pivot will increase a single non-basic variable (corresponding to a column with $c_j > 0$), keeping the other non-basic variables set to zero. We adjust the current basic variables to keep the equations satisfied. Notice that our right-hand side is always non-negative, so each solution will be *feasible*.[1] This is ensured by the way we do our pivots: the numbers that we compute and write to the right of the tableau tell us how large the entering variable can grow before some basic variable must be adjusted to a negative value. When we pick the smallest such bound to choose the leaving variable, we ensure sure that all basic variables stay non-negative. If you are performing simplex pivots and find a negative number on the right-hand side, then you have made a mistake somewhere!

If we look at the tableaux, we see that each gives us a solution (x_1, x_2) to our original program. Remembering that any variables not listed on the left have value zero, we obtain the sequence of solutions

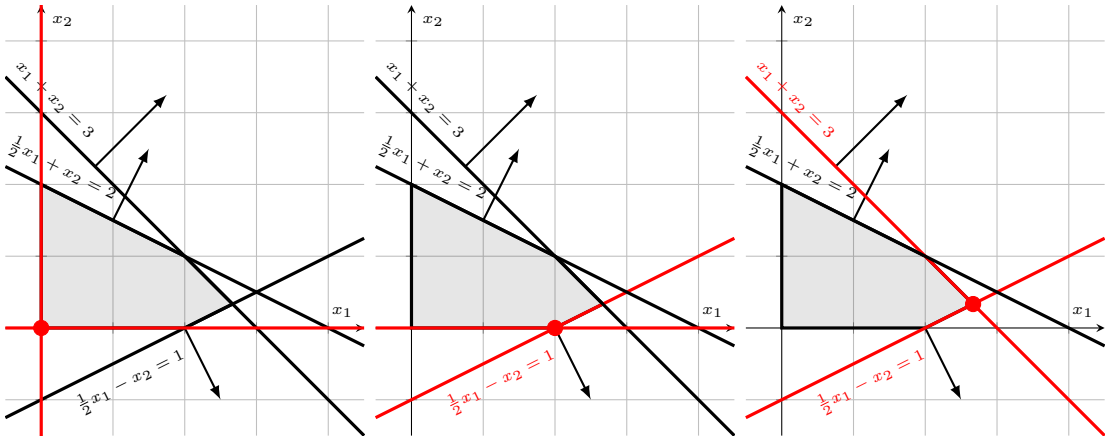
$$(0,0), (2,0), \left(\frac{8}{3}, \frac{1}{3}\right).$$

Each non-basic variables in one of our tableaux will correspond to a set of tight constraints or restrictions:

- In our first tableau, we have x_1 and x_2 non-basic, so the restrictions $x_1 \geq 0$ and $x_2 \geq 0$ should be tight.
- In our second tableau, we have x_2 and s_3 non-basic, so the restriction $x_2 \geq 0$ and the constraint $\frac{1}{2}x_1 - x_2 \leq 1$ corresponding to the slack variable s_3 will be tight.
- In our final tableau, we have s_1 and s_3 non-basic, so the constraints $x_1 + x_2 \leq 3$ and $\frac{1}{2}x_1 - x_2 \leq 1$ corresponding to these slack variables will be tight.

Plotting the points and tight constraints, we find the following sequence of pictures:

[1] In fact, it can be shown (but here we will take for granted) that the algorithm always gives us a set of basic variables whose columns are linearly independent, so these will all be *basic feasible solutions*.



We see that, indeed, all of the points are feasible solutions. Each pivot adjusts the set of tight constraints defining the solution: we drop one tight constraint (corresponding to the leaving variable) and replace it with another tight constraint (corresponding to the entering variable). This has the effect of moving our solution along some "edge" of the feasible region. Each such move will be along an edge that increases the objective. Here, our objective points in the direction $\mathbf{c}^\mathsf{T} = (4, 1/2)$ and we can see that we move along edges that increase our projection onto this vector.

6.2 Finding an Initial Feasible Solution

So far, we have only applied the simplex algorithm to a problems of a very specific type, where we could simply take all of our slack variables as basic variables to construct the initial tableau. For this to work we needed 2 things to be true when we rewrote our problem into standard equation form. First, every equation needed to have a slack variable. Second, the right-hand side of all these equations had to be non-negative. If the first condition did not hold, we would not obtain a full set of m linearly independent columns from our slack variables. If the second condition did not hold, our initial basic feasible solution would set a basic variable to some negative value b_i (if we set all non-basic variables to zero, as usual). We now show how to handle both of these problems together. Along the way, we will also see how to detect *infeasible* linear programs.

As usual, let's adopt the convention that our slack variables are named $s_1, s_2, \ldots$. Of course, the simplex algorithm does not care whether a variable is a slack variable or not, so this is just a naming convention to help us keep things clear when thinking about it and doing calculations by hand. Let's see an example of a (small)

linear program that we cannot handle with our existing simplex algorithm:

$$\begin{aligned} \text{maximize} \quad & 10x_1 + 15x_2 + 8x_3 \\ \text{subject to} \quad & 8x_1 + 6x_2 + 12x_3 \leq 24 \\ & 4x_1 + 6x_2 + 6x_3 \geq 6 \\ & 6x_1 + 4x_2 + 8x_3 = 12 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Converting directly to standard equation form, we get:

$$\begin{aligned} \text{maximize} \quad & 10x_1 + 15x_2 + 8x_3 \\ \text{subject to} \quad & 8x_1 + 6x_2 + 12x_3 + s_1 = 24 \\ & -4x_1 - 6x_2 - 6x_3 + s_2 = -6 \\ & 6x_1 + 4x_2 + 8x_3 = 12 \\ & x_1, x_2, x_3, s_1, s_2 \geq 0 \end{aligned}$$

Here, we first flipped our $\geq$ inequality to be $\leq$ (multiplying both sides by -1) then added slack variables to each inequality. Now, we have a linear program in standard equation form. If we could just find an initial basi feasible solution, we could apply the simplex algorithm. However, it's not so clear how to do this. We need to find a set of 3 variables, corresponding to linearly independent columns, so that when we set all of our other variables to zero and solve we get a solution that has no basic variable taking a negative value. Notice that this is probably not that much easier than just solving the problem, unfortunately.

Before, we were just starting by taking each slack variable as a basic variable. The system of equations was then easy to solve: each slack variable occurred in exactly one equation, and so it's value in the corresponding basic feasible solution was equal to the right-hand side. Now, we cannot do this: the second equation would set $s_2 = -6$, which is not a feasible solution, and the third equation doesn't even *have* a slack variable! How should we find an initial basic feasible solution to start the algorithm? The answer is that we can run the Simplex algorithm on a slightly modified problem to find a feasible solution, then start our regular Simplex algorithm from there.

In order to solve the above problem, we will (temporarily) introduce new *artificial* variables into the linear program. Our goal is that each equation should have either a slack variable or an artificial variable that can be chosen to get a initial basic feasible solution. This is already the case for the first equation, so let's leave it alone. We solve our problem with the second equation by *subtracting* a non-negative artificial variable a_1 from the right hand side. Then, if we select a_1

as a basic variable and leave all other variables in this equation non-basic, we will get $-a_1 = -6$ which is the same as setting $a_1 = 6$. For the last equation, we don't have a slack variable so, again, let's introduce a non-negative artificial variable a_2. Since the right-hand side is positive, we *add* a_2 to the left hand side. Again, if all the other variables in this equation are non-basic, we will get $a_2 = 12$, which is a feasible solution. Our final system of equations looks like:

$$\begin{array}{rcrcrcrcrcrcrcl} 8x_1 & + & 6x_2 & + & 12x_3 & + & s_1 & & & & & & & = & 24 \\ -4x_1 & - & 6x_2 & - & 6x_3 & & & + & s_2 & - & a_1 & & & = & -6 \\ 6x_1 & + & 4x_2 & + & 8x_3 & & & & & & & + & a_2 & = & 12 \end{array}$$

Now, it is relatively clear that we can select s_1, a_1, a_2 as a valid basic feasible solution. But obviously this is not going to model our original set of constraints! If $a_2 > 0$, then we will have:

$$6x_1 + 4x_2 + 8x_3 < 12$$

but the original linear program asked for:

$$6x_1 + 4x_2 + 8x_3 = 12\,.$$

Similarly, if $a_1 > 0$, then we will have:

$$-4x_1 - 6x_2 - 6x_3 + s_2 > -6\,.$$

If we set s_2 to 0 as well, we end up violating our constraint, since then:

$$-4x_1 - 6x_2 - 6x_3 \leq -6\,.$$

Intuitively a_1 and a_2 now represent *how much we violate* each constraint they appear in. If a_1 and a_2 were both set to 0, we would recover our original standard equation form program. One way to do this is to make $a_1 + a_2$ as small as possible: since a_1 and a_2 are both restricted to be non-negative, we can have $a_1 + a_2 = 0$ if and only if both $a_1 = 0$ and $a_2 = 0$. We can formulate this goal as a linear program:

$$\begin{array}{ll} \text{minimize} & a_1 + a_2 \\ \text{subject to} & 8x_1 + 6x_2 + 12x_3 + s_1 = 24 \\ & -4x_1 - 6x_2 - 6x_3 + s_2 - a_1 = -6 \\ & 6x_1 + 4x_2 + 8x_3 + a_2 = 12 \\ & x_1, x_2, x_3, s_1, s_2, a_1, a_2 \geq 0 \end{array}$$

We can rewrite this as a maximisation problem by simply negating the objective as usual:

$$\begin{aligned}
\text{maximize} \quad & -a_1 - a_2 \\
\text{subject to} \quad & 8x_1 + 6x_2 + 12x_3 + s_1 = 24 \\
& -4x_1 - 6x_2 - 6x_3 + s_2 - a_1 = -6 \\
& 6x_1 + 4x_2 + 8x_3 + a_2 = 12 \\
& x_1, x_2, x_3, s_1, s_2, a_1, a_2 \geq 0
\end{aligned}$$

Now, as we have seen, we can easily construct an initial basic feasible solution to our set of constraints: for each equation with a positive right-hand side and a slack variable, we take the slack variable as basic. For each other equation, we use the artificial variable that we introduced. In tableau form, we get something like this:

	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
s_1	8	6	12	1	0	0	0	24
a_1	-4	-6	-6	0	1	-1	0	-6
a_2	6	4	8	0	0	0	1	12
	0	0	0	0	0	-1	-1	0

Notice that here our objective is to maximise $-a_1 - a_2$. Let's call this new objective w to distinguish it from the original problem's objective. It will be convenient to carry along the original problem's objective, as well, so let's list it below the line, too. We will write $-w$ and $-z$ to the left of our tableau to remind ourselves which is which (this minus sign is to help us remember that the right-hand side is -1 times the current value of these objectives). We will ignore the row for z when choosing our pivot elements, but we will update it in each pivot step as usual (that is, we will always update the z row so that its non-zero coefficients are all non-basic variables). Let's also add a row at the top of the tableau to remind us of which column goes with which variable. Altogether we get:

	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
s_1	8	6	12	1	0	0	0	24
a_1	-4	-6	-6	0	1	-1	0	-6
a_2	6	4	8	0	0	0	1	12
$-w$	0	0	0	0	0	-1	-1	0
$-z$	10	15	8	0	0	0	0	0

We are not quite ready to start yet. We need to make sure that the objective function w is written only in terms of *non-basic* variables, and we want each column corresponding to a basic variable to have an entry of 1 for that variable an 0 for all other variables. Both our easy to solve. First, we simply multiply the row for a_1 by -1:

	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
s_1	8	6	12	1	0	0	0	24
a_1	4	6	6	0	-1	1	0	6
a_2	6	4	8	0	0	0	1	12
$-w$	0	0	0	0	0	-1	-1	0
$-z$	10	15	8	0	0	0	0	0

Then, we add the row for a_1 and the row from a_2 from the row for $-w$ to zero out its entries for a_1 and a_2:

	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
s_1	8	6	12	1	0	0	0	24
a_1	4	6	6	0	-1	1	0	6
a_2	6	4	8	0	0	0	1	12
$-w$	10	10	14	0	-1	0	0	18
$-z$	10	15	8	0	0	0	0	0

Now, we are all set up for the simplex algorithm. Remember: we are now trying to maximise w, so we will choose our entering variables by looking in its row. The largest positive coefficient in this row is 14, so x_3 will enter. We get bounds of 2, 1 and 3/2 for our tableau rows. The second row's bound is smallest, so a_1 is the leaving variable. We carry out the pivot and get:

	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
s_1	0	-6	0	1	2	-2	0	12
x_3	$\frac{2}{3}$	1	1	0	$-\frac{1}{6}$	$\frac{1}{6}$	0	1
a_2	$\frac{2}{3}$	-4	0	0	$\frac{4}{3}$	$-\frac{4}{3}$	1	4
$-w$	$\frac{2}{3}$	-4	0	0	$\frac{4}{3}$	$-\frac{7}{3}$	0	4
$-z$	$\frac{14}{3}$	7	0	0	$\frac{4}{3}$	$-\frac{4}{3}$	0	-8

Now, we have column 5 as the pivot column, since 4/3 is the largest value in our row for w. Thus, s_2 is the entering variable. We get a bound of 6 for the first row, no bound for the second row, and 3 for the third row. So, our third row is our pivot row and our entering variable is a_2. Carrying out the pivot gives us:

	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
s_1	-1	0	0	1	0	0	$-\frac{3}{2}$	6
x_3	$\frac{3}{4}$	$\frac{1}{2}$	1	0	0	0	$\frac{1}{8}$	$\frac{3}{2}$
s_2	$\frac{1}{2}$	-3	0	0	1	-1	$\frac{3}{4}$	3
$-w$	0	0	0	0	0	-1	-1	0
$-z$	4	11	0	0	0	0	-1	-12

At this point, we see that there is no choice for entering variable, so we are at an optimal solution. The value of $-w = 0$, and so this solution must set all artificial variables to zero. Indeed, we find that all artificial variables are non-basic. The values of other variables now give a feasible solution to our problem. We now start another phase of the simplex algorithm starting from this solution. In this phase we will keep a_1 and a_2 fixed to zero. The easiest way to do this is to just drop the row for $-w$ and the columns for a_1 and a_2 from the tableau. Our first tableau for the second phase of simplex is:

	x_1	x_2	x_3	s_1	s_2	
s_1	-1	0	0	1	0	6
x_3	$\frac{3}{4}$	$\frac{1}{2}$	1	0	0	$\frac{3}{2}$
s_2	$\frac{1}{2}$	-3	0	0	1	3
$-z$	4	11	0	0	0	-12

Now, we proceed as usual, using our $-z$ row to find pivots. In the next step, we will have x_2 entering and x_3 leaving. The pivot gives:

	x_1	x_2	x_3	s_1	s_2	
s_1	-1	0	0	1	0	6
x_2	$\frac{3}{2}$	1	2	0	0	3
s_2	5	0	6	0	1	12
$-z$	$-\frac{25}{2}$	0	-22	0	0	-45

Now, we are finished. We have found an optimal solution of our original linear program corresponding to $x_1 = 0$, $x_2 = 3$, $x_3 = 0$. The value of this solution is 45.

6.3 The 2-Phase Simplex Algorithm

The algorithm we used in our example is called the *2-phase simplex algorithm*. As we saw, it works in 2 phases: the first phase introduces artificial variables to find an "obvious" initial basic feasible solution. Then, it executes the simplex algorithm, to try and maximise an artificial objective function that is equal to -1 times the sum of all these artificial variables. If it finds a solution of value 0, then all these artificial variables must be 0 and we have found a solution to our original program. We drop all the artificial variables and then continue from this solution with the simplex algorithm, this time maximising our original objective function. If the first phase terminates at a solution with value *less than* zero, the linear program must be infeasible. We know that when the simplex algorithm terminates, it has found an optimal solution and if our original linear program has any feasible solution, we could simply use the values of the variables in this solution to construct a feasible solution of our extended program, by setting all of artificial variables set to zero. This would give us a feasible solution with artificial objective value 0. Thus, the only way the first phase can terminate with an optimal, artificial objective less than 0, is if there is *no* feasible solution at all. What if our initial phase gives us an unbounded program? This cannot happen, since we are maximising -1 times the sum of a set of non-negative variables. Thus, our artificial objective function will always be less than 0, so our phase 1 program is trivially bounded.

The general algorithm is shown on the next page.

Algorithm 6.1 (2-Phase Simplex Algorithm)**.** Given any linear program, the 2-phase simplex algorithm works as follows:

1. Rewrite the linear program into standard equation form, introducing slack variables, just as in the standard simplex algorithm.
2. For each equation in your program, check to see if: (1) it has a slack variable and (2) its right-hand side is non-negative. If either of these conditions fail, we introduce a new, non-negative, artificial variable for this equation. We add it to the left-hand side with a sign equal to the sign of the right-hand side (i.e. if the right-hand side is negative, its coefficient is -1, otherwise its coefficient is 1).
3. We form an initial basic feasible solution as follows: for each equation with an artificial variable, choose that artificial variable to be basic. Each equation without an artificial variable should have a slack variable that we can choose to be basic.
4. We now construct our tableau, using these variables as our basic variables. Below the line, we write 2 objective functions: first, we write a row for our phase 1 objective w. This is -1 in all columns corresponding to artificial variables, and 0 everywhere else. Below this we copy the value of the objective z, just as in the standard simplex algorithm.
5. Now, we need to clean up our tableau. If any artificial variable has a -1 entry in its corresponding row and column, we multiply its row by -1. After doing this, we add each artificial variable's row to the row for w. This should result in w being non-zero only for *non-basic* variables.
6. We now proceed as in the standard simplex algorithm, but use our w row as the objective when choosing our entering variables.
7. If, when the algorithm terminates, the final tableau gives $w > 0$, then the original linear program is *infeasible*—stop.
8. Otherwise, we must have all artificial variables set to zero. If any artificial variable is currently basic, we have a *degenerate solution* setting this variable to zero and we need to perform an extra step, which we will describe separately. At the end we have a tableau with all artificial variables non-basic.
9. Delete the columns for all artificial variables and the row for w from the tableau.
10. Continue with the standard simplex algorithm from the current tableau, using the z to find the entering variables.

As we have noted, if the first phase of simplex gives us a solution with objective value $w = 0$, then all artificial variables must be set to zero and so our non-artificial variables will satisfy our problem's constraints. In step 8, we drop all artificial variables from the tableau. This is effectively prevents them from ever becoming basic again, and so is like insisting that they stay set to zero for the rest of the algorithm.

6.3.1 Driving Artificial Variables Out of the Basis (non-examinable)

One small difficulty is that if we have a *degenerate* solution, some artificial variable may be set to zero but still be basic. In this case, we cannot directly continue to phase 2, but first need to carry out a special step to drive these artificial variables out of the basis. **Note:** This won't happen on any examples in your coursework or the examination, so this procedure is only here for the sake of completeness.

Let's now describe how this is done:

Algorithm 6.2 (Driving Artificial Variables Out of the Basis)**.** Suppose that at the end of phase 1 of the 2-phase simplex algorithm, we have a tableau with an artificial variable a_i that is basic. We drive this variable out of the basis as follows:

1. Look at the row for the basic artificial variable a_i and find the first non-zero entry in this row that corresponds to a non-artificial variable.
2. Perform a pivot operation on this column, treating this non-artificial variable as the entering variable and a_i as the leaving variable. As usual, we update the right-hand side and the equation for z as part of this pivot. Afterwards, a_i will be non-basic.

By repeating the above procedure for each artificial variable that is basic, we can arrive at a tableau that has all artificial variables non-basic. We can then easily get rid of the artificial variables as described in step 8 of the 2-phase simplex algorithm.

You may be wondering why it is okay to perform these pivots to drive out artificial variables. Note that we *do not* check to make sure the non-artificial variable has a positive value in our row for z. Performing row operations will preserve the validity of all of our equations, but this step might decrease our objective or make our solution infeasible by changing some other basic variable's value. Luckily, neither of these things can happen. Indeed, remember that in this case our artificial variable must be set to 0 at the current solution, which means

that the right-hand side of its row in the tableau is 0. This means that when we carry out row operations, we will not alter the current value of any other basic variable *or* the current value of the objective function (since we will just add some multiple of 0 to the right hand side of each row).

We'll return to the topic of degeneracy briefly in the next lecture. For now, let's see an example of how we can drive out artificial variables.

Example 6.1. Solve the following linear program:

$$\begin{aligned} \text{maximize} \quad & 4x_1 + 3x_2 \\ \text{subject to} \quad & 3x_1 + 6x_2 = 6 \\ & 2x_1 - 2x_2 \geq 4 \end{aligned}$$

Solution. After converting to standard equation form, and adding slack variables, our constraints look like:

$$\begin{array}{rcrcrcr} 3x_1 & + & 6x_2 & & & = & 6 \\ -2x_1 & + & 2x_2 & + & s_1 & = & -4 \end{array}$$

We don't have any slack variable in the first equation, and our second equation has a negative right-hand side, so we need to add artificial variables to both. Following the rules for adding them, we get:

$$\begin{array}{rcrcrcrcr} 3x_1 & + & 6x_2 & & + a_1 & & & = & 6 \\ -2x_1 & + & 2x_2 & + & s_1 & - & a_2 & = & -4 \end{array}$$

We then write down our initial tableau and carry out the steps to make it valid. Our first valid tableau is:

	x_1	x_2	s_1	a_1	a_2	
a_1	3	6	0	1	0	6
a_2	2	-2	-1	0	1	4
$-w$	5	4	-1	0	0	10
$-z$	4	3	0	0	0	0

After our first pivot, we get:

	x_1	x_2	s_1	a_1	a_2	
x_1	1	2	0	$\frac{1}{3}$	0	2
a_2	0	-6	-1	$-\frac{2}{3}$	1	0
$-w$	0	-6	-1	$-\frac{5}{3}$	0	0
$-z$	0	-5	0	$-\frac{4}{3}$	0	-8

This tableau represents an optimal solution to the first phase. The solution has $w = 0$, so the program is feasible. However, we have an artificial variable (namely, a_2) that is basic. To fix this, we can find any other non-artificial variable with a non-zero value in the row for a_2. Let's pick x_2. We perform a pivot operation treating x_2 as the entering variable and a_2 as the leaving variable. Notice that we do not do our normal checks on the objective function or compute our bounds—we just execute the pivot. We get:

	x_1	x_2	s_1	a_1	a_2	
x_1	1	0	$-\frac{1}{3}$	$\frac{1}{9}$	$\frac{1}{3}$	2
x_2	0	1	$\frac{1}{6}$	$\frac{1}{9}$	$-\frac{1}{6}$	0
$-w$	0	0	0	-1	-1	0
$-z$	0	0	$\frac{5}{6}$	$-\frac{7}{9}$	$-\frac{5}{6}$	-8

Our objective value and the values of all variable stayed the same, as we expected. Now, all artificial variables are non-basic, so we can drop their columns, as well as the row for $-w$. Our starting tableau for the second phase will be:

	x_1	x_2	s_1	
x_1	1	0	$-\frac{1}{3}$	2
x_2	0	1	$\frac{1}{6}$	0
$-z$	0	0	$\frac{5}{6}$	-8

We now run the simplex algorithm starting from this tableau. Our first pivot (remember, now we use the z row) has s_1 entering and x_2 leaving. We get:

	x_1	x_2	s_1	
x_1	1	2	0	2
s_1	0	6	1	0
$-z$	0	-5	0	-8

Here, the second phase terminates since the z row has no positive entry. The optimal solution of our original program sets $x_1 = 2$ and $x_2 = 0$. □

6.4 Termination of the Simplex Algorithm

We have seen that the simplex algorithm is capable of identifying unbounded programs, and that the first phase of the 2-phase simplex algorithm will identify infeasible programs. We have also seen that if the algorithm terminates, the final tableau represents an optimal solution. Here, we consider one last question: how do we know that the algorithm always terminates? We have seen some examples in the last lecture in which some pivots might not increase the objective function. In these steps, the row for leaving variable must have a zero on the right side of the tableau, since otherwise the row operations we performed on the last line would change the bottom-right entry. This means that when we pivot on this row, the entire right side of the tableau stays the same. In other words, we stay at exactly the same solution. Let's see an example of this happening to two dimensions to get a better idea of what's going on.

Consider the following linear program in 2 variables:

$$\begin{aligned} \text{maximize} \quad & x_2 \\ \text{subject to} \quad & -2x_1 + x_2 \leq 1 \\ & -\frac{3}{2}x_1 + x_2 \leq 3 \\ & -x_1 + x_2 \leq 5 \\ & -\frac{3}{4}x_1 + x_2 \leq 6 \\ & x_1 \leq 10 \\ & x_2 \leq 10 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{6.3}$$

When we convert to standard inequality form, we will add slack variables $s_1, \ldots, s_6$. Figure 6.1 shows a picture of the feasible region, where we have omitted the normal vectors of each constraint to avoid clutter, and labelled each constraint by its slack variable.

The simplex algorithm we have learned will generate the following sequence of tableaux shown in Figure 6.2. The third and fourth pivot steps here are degenerate, and we can see what that they have in common: our leaving variable was *already* set to zero before we carried out the pivot. For example, in both the second, third, and fourth tableaux, we had 2 basic variables set to zero. In general, if a basic

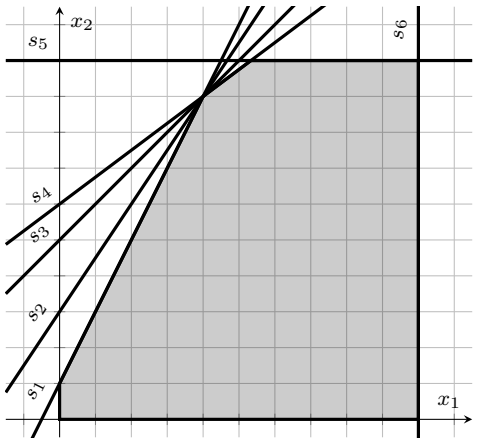


Figure 6.1: Plot of the feasible region for program (6.3)

feasible solution sets a basic variable to zero, we call the solution *degenerate*. We have seen that the zero variables correspond to tight constraints of our original linear program, so it makes sense to expect the solutions corresponding to these tableaux to have *more than* 2 tight constraints. Indeed, we should expect them to have 4: 2 for the non-basic variables, and 2 for the 2 basic variables that are set to zero. If we plot the solutions on our picture, highlighting the non-basic constraints with red lines, we see the sequence if plots shown in Figure 6.3.

	x_1	x_2	s_1	s_2	s_3	s_4	s_5	s_6	
s_1	-2	1	1	0	0	0	0	0	1
s_2	$-\frac{3}{2}$	1	0	1	0	0	0	0	3
s_3	-1	1	0	0	1	0	0	0	5
s_4	$-\frac{3}{4}$	1	0	0	0	1	0	0	6
s_5	0	1	0	0	0	0	1	0	10
s_6	1	0	0	0	0	0	0	1	10
$-z$	0	1	0	0	0	0	0	0	0

$\longrightarrow$

	x_1	x_2	s_1	s_2	s_3	s_4	s_5	s_6	
x_2	-2	1	1	0	0	0	0	0	1
s_2	$\frac{1}{2}$	0	-1	1	0	0	0	0	2
s_3	1	0	-1	0	1	0	0	0	4
s_4	$\frac{5}{4}$	0	-1	0	0	1	0	0	5
s_5	2	0	-1	0	0	0	1	0	9
s_6	1	0	0	0	0	0	0	1	10
$-z$	2	0	-1	0	0	0	0	0	-1

$\swarrow$

	x_1	x_2	s_1	s_2	s_3	s_4	s_5	s_6	
x_2	0	1	-3	4	0	0	0	0	9
x_1	1	0	-2	2	0	0	0	0	4
s_3	0	0	1	-2	1	0	0	0	0
s_4	0	0	$\frac{3}{2}$	$-\frac{5}{2}$	0	1	0	0	0
s_5	0	0	3	-4	0	0	1	0	1
s_6	0	0	2	-2	0	0	0	1	6
$-z$	0	0	3	-4	0	0	0	0	-9

$\longrightarrow$

	x_1	x_2	s_1	s_2	s_3	s_4	s_5	s_6	
x_2	0	1	0	-2	3	0	0	0	9
x_1	1	0	0	-2	2	0	0	0	4
s_1	0	0	1	-2	1	0	0	0	0
s_4	0	0	0	$\frac{1}{2}$	$-\frac{3}{2}$	1	0	0	0
s_5	0	0	0	2	-3	0	1	0	1
s_6	0	0	0	2	-2	0	0	1	6
$-z$	0	0	0	2	-3	0	0	0	-9

$\swarrow$

	x_1	x_2	s_1	s_2	s_3	s_4	s_5	s_6	
x_2	0	1	0	0	-3	4	0	0	9
x_1	1	0	0	0	-4	4	0	0	4
s_1	0	0	1	0	-5	4	0	0	0
s_2	0	0	0	1	-3	2	0	0	0
s_5	0	0	0	0	3	-4	1	0	1
s_6	0	0	0	0	4	-4	0	1	6
$-z$	0	0	0	0	3	-4	0	0	-9

$\longrightarrow$

	x_1	x_2	s_1	s_2	s_3	s_4	s_5	s_6	
x_2	0	1	0	0	0	0	1	0	10
x_1	1	0	0	0	0	$-\frac{4}{3}$	$\frac{4}{3}$	0	$\frac{16}{3}$
s_1	0	0	1	0	0	$-\frac{8}{3}$	$\frac{5}{3}$	0	$\frac{5}{3}$
s_2	0	0	0	1	0	-2	1	0	1
s_3	0	0	0	0	1	$-\frac{4}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$
s_6	0	0	0	0	0	$\frac{4}{3}$	$-\frac{4}{3}$	1	$\frac{14}{3}$
$-z$	0	0	0	0	0	0	-1	0	-10

Figure 6.2: Tableaux generated for program (6.3)

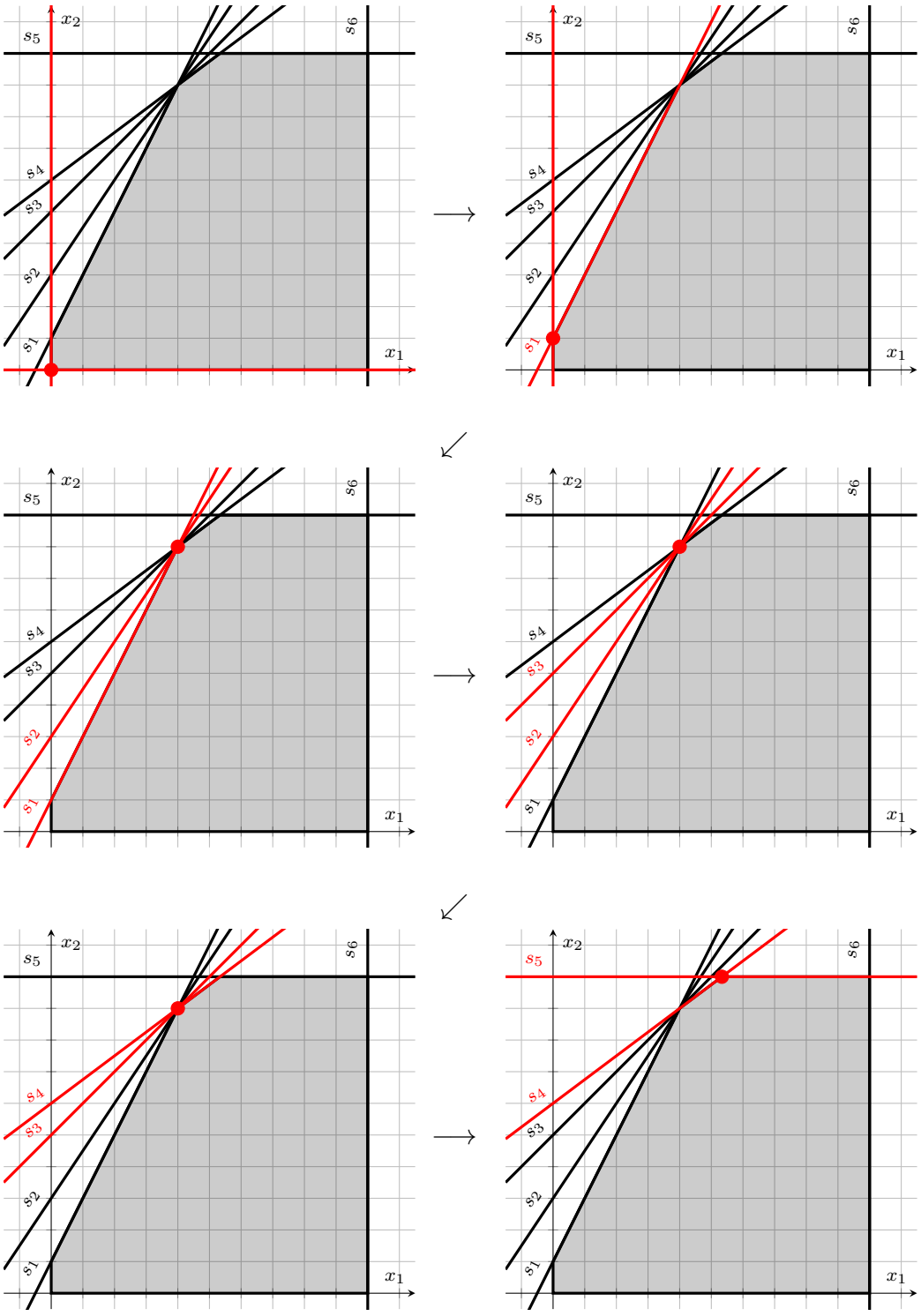


Figure 6.3: Plots of steps for the simplex algorithm applied to problem (6.3)

Indeed, in each of our degenerate feasible solutions (which correspond to the point $(4, 9)$) we have 4 tight constraints but only 2 correspond to non-basic variables. Our degenerate pivots swap one of these constraints for another and so does not change the solution or the objective, but only changes which of the constraints we currently consider as a *non-basic* variable. Eventually, we make progress, but is this always the case? Consider the following initial tableau and sequence of pivots.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	0.5	−5.5	−2.5	9	1	0	0	0
x_6	0.5	−1.5	−0.5	1	0	1	0	0
x_7	1	0	0	0	0	0	1	1
$-z$	10	−57	−9	−24	0	0	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	1	−11	−5	18	2	0	0	0
x_6	0	4	2	−8	−1	1	0	0
x_7	0	11	5	−18	−2	0	1	1
$-z$	0	53	41	−204	−20	0	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	1	0	0.5	−4	−0.75	2.75	0	0
x_2	0	1	0.5	−2	−0.25	0.25	0	0
x_7	0	0	−0.5	4	0.75	−2.75	1	1
$-z$	0	0	14.5	−98	−6.75	−13.25	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_3	2	0	1	−8	−1.5	5.5	0	0
x_2	−1	1	0	2	0.5	−2.5	0	0
x_7	1	0	0	0	0	0	1	1
$-z$	−29	0	0	18	15	−93	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_3	−2	4	1	0	0.5	−4.5	0	0
x_4	−0.5	0.5	0	1	0.25	−1.25	0	0
x_7	1	0	0	0	0	0	1	1
$-z$	−20	−9	0	0	10.5	−70.5	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	-4	8	2	0	1	-9	0	0
x_4	0.5	-1.5	-0.5	1	0	1	0	0
x_7	1	0	0	0	0	0	1	1
$-z$	22	-93	-21	0	0	24	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	0.5	-5.5	-2.5	9	1	0	0	0
x_6	0.5	-1.5	-0.5	1	0	1	0	0
x_7	1	0	0	0	0	0	1	1
$-z$	10	-57	-9	-24	0	0	0	0

Notice that this sequence ends with the same tableau we started with. This is bad, because it means the simplex algorithm will run forever, carrying out the same sequence of pivots again and again! We call this phenomenon *cycling*, since the algorithm cycles back to where it started. One can argue that in practice such problems are unlikely to occur. Nevertheless, it will be useful in later proofs to show that the simplex algorithm always finds an optimal tableau for any problem that is not infeasible or unbounded. To argue this, we need to make sure our algorithm always eventually reaches this tableau!

6.5 Alternative Pivot Rules (non-examinable)

One way of handling this problem is to change the rule we use to find entering and leaving variables in each round. We have been selecting our entering variable by choosing the *largest positive* coefficient in the last row of the tableau. In principle, however, choosing any positive coefficient would be okay—as we saw in the equation form of the simplex algorithm, as long as a variable has a positive coefficient, increasing it should increase the objective.

Suppose we have fixed an ordering of all of our variables. If all of our variables are named x_i, we can just order them naturally in increasing order of i. For simplicity let's suppose that this is the case. The following pivot rule uses this ordering to select which variable enters and which leaves. Note that we still do not have any variable with a non-positive coefficient in the z row entering, and we always select some variable with the smallest bound to leave. This ensures that our algorithm remains correct.

Definition 6.1 (Bland's Rule)**.** At each step of the simplex algorithm, *Bland's Rule* examines all of the variables x_i whose entry in the z row is positive, and selects from these the one with the *smallest subscript* i to be the entering variable. It then computes a bound for each row as usual, and if several rows have the same smallest bound, it selects from these the one whose corresponding basic variable x_i has the smallest subscript i and sets this x_i to be the leaving variable.

You can think about Bland's Rule as simply being a form of tie breaking. If several variables all have positive entry in the z row, it chooses the one with the smallest subscript, instead of choosing the one with the largest entry. Similarly, if several variables could be the leaving variable, it breaks the tie by choosing the one with the smallest subscript. One can show the following:

Theorem 6.1. If Bland's Rule is used to select the entering and leaving variables in each pivot operation of the simplex algorithm, then the algorithm will never cycle.

Proof. **This Proof is Optional and Non-Examinable.** Suppose (for the sake of contradiction) that the simplex algorithm using Bland's pivoting rule performs a set of degenerate pivots, producing a sequence of tableaux $T_0, T_1, \ldots, T_k, T_0$ that returns to where it started. Let F denote the set of variables that are entering or leaving variables in one of these degenerate steps. Note that every variable that leaves must also enter at some point (and vice versa) since we must somehow end up back where we started. Let x_t be the variable of F that has the largest subscript. There must be some pivot where x_t is the leaving variable and some other variable (let's call it x_s) is the entering variable. Call the tableau immediately before this pivot T and let B be the set of indices of the basic variables in T (that is, x_j is basic in T if and only if $j \in B$). Then, we must have the rightmost value $b_j = 0$ for every $j \in F$.

For each $i \in B$, the tableau T gives us an equation:

$$x_i + \sum_{j \notin B} a_{ij} x_j = b_j \tag{6.4}$$

Also, we have an equation for z, reading:

$$z = \sum_{j \notin B} c_j x_j + d \tag{6.5}$$

where the values in the bottom row are c_i and the bottom right corner of the tableau is $-d$. Note that since x_s was the entering variable, we must have $c_s > 0$,

and also $a_{st} > 0$. Also, since x_t was chosen to have the largest index in F, we must be $s < t$.

Let's construct a solution satisfying our equations (6.4) as follows (note that the solution we construct will not necessarily be feasible! All that we will require is that it satisfies the equations (6.4)). First, set $x_s = \theta$ for any constant θ. For all other $j \notin B$, we set $x_j = 0$. Finally, for all $i \in B$, we set $x_i = b_i - a_{is}\theta$. Then, for every $i \in B$, the right-hand side of (6.4):

$$x_i + \sum_{j \notin B} a_{ij} x_j = (b_i - a_{is}\theta) + a_{is}\theta = b_i$$

as required. For this solution, our equation for z gives us:

$$z = \sum_{j \notin B} c_j x_j + d = c_s \theta + d \,.$$

Somewhere else in our "cycle" of tableaux x_t must be the entering variable. Call the tableau right before this pivot T'. Then, if we set c'_j T' also gives us an equation for z. Let's write it as:

$$z = \sum_j c'_j x_j + d \tag{6.6}$$

Here, we have simply sum over all the variables, setting $c'_j = 0$ for any j that is not basic in T'. Notice that since all of the pivots in our cycle are degenerate, we must have the values of d and z the same throughout. Our pivots simply rearrange the equations in our tableau, and so any solution of (6.4) and (6.5) will also satisfy (6.8). Plugging our previously constructed solution into (6.6), we get:

$$z = \sum_j c'_j x_j + d = c'_s \theta + \sum_{i \in B} c'_i (b_i - a_{is}\theta) \,, \tag{6.7}$$

since the only non-zero variables are x_s and the variables x_j for $j \in B$. Since both (6.7) and (6.5) are valid, we must have:

$$c_s \theta + d = c'_s \theta + \sum_{i \in B} c'_i (b_i - a_{is}\theta) \,,$$

which we rearrange as:

$$\left(c_s - c'_s + \sum_{i \in B} c'_i a_{is} \right) \theta = \sum_{i \in B} c'_i b_i \,.$$

This holds *regardless* of how we choose θ. So, for every possible choice of θ, the right-hand side must equal the left-hand side, which is a *constant* independent of

θ. This can happen only if:

$$\left(c_s - c'_s + \sum_{i \in B} c'_i a_{is} \right) = 0 \,. \tag{6.8}$$

But, we have $c_s > 0$, since s entered in the pivot for tableau T. Now, $s < t$ but t is entering in pivot for tableau T'. Since Bland's rule chooses the eligible variable with the smallest index, it must be the case that x_s was not eligible in this pivot, and so $c_{s'} \leq 0$. But, this means that $c_s - c'_s > 0$. Thus, (6.8) implies that we must have $\sum_{i \in B} c'_i a_{is} < 0$, and so $c'_i a_{is} < 0$ for some $i \in B$.

Let $r \in B$ be one such index with $c'_r a_{rs} < 0$. Now, since $r \in B$, x_r is basic in tableau T. But, if $c'_r a_{rs} < 0$, then $c'_r \neq 0$, so r must be *non-basic* in tableau T'. In other words, $r \in F$ (the set of variables that change between basic and non-basic in our cycle) and since t the largest index of any variable in F, we have $r \leq t$. We now show that in fact $r < t$. Indeed, as we have already noted, $a_{it} > 0$, since t is the leaving variable in the pivot from tableau T, and, since t is the entering variable in the pivot from tableau T', $c'_t > 0$. But, r has $c'_r a_{rs} < 0$. Thus, $r \neq t$.

Now, $r < t$ and x_r and x_t are both non-basic in tableau T', but t is the entering variable in the pivot from this tableau. Again, if r were eligible to enter in this pivot, Bland's rule would have selected it instead of t, so we must have $c'_r \leq 0$. Since $c'_r a_{rs} < 0$, this implies that $a_{rs} > 0$. Also, since $r \in F$, we have $b_r = 0$. But this means that we must have recorded a 0 for row r when selected the leaving variable from tableau T. Since $r < t$, we should thus have selected r as the leaving variable in T rather than t—a contradiction. □

In practice, many linear programming solvers just ignore the problem of degeneracy, for the following reasons. First, using Bland's Rule doesn't give us any choice for entering and leaving variables, and some other methods have been shown to usually increase the objective faster. Second, if we look at our picture of degeneracy, we find that it requires *several* lines to intersect in a single point. When we implement simplex on a computer, we use *floating-point* numbers, which are decimals with only a fixed amount of precision. This introduces small errors into the arithmetic because we always need to round off all but the first few decimals to store our numbers. The effect of these errors is another potential cause for worry (which we will not consider here), but a happy side-effect is that they mean it is unlikely that several equations go through *exactly* the same point, because this requires that all of their coefficients be *exactly* correct.

Week 7

Duality (I)

Now we turn to our next major topic, which is *duality*. We begin with a small motivating example. Consider the following linear program:

$$\begin{aligned} \text{maximize} \quad & 2x_1 + 3x_2 + x_3 \\ \text{subject to} \quad & x_1 + x_2 + x_3 \leq 10 \\ & \tfrac{1}{2}x_1 + x_2 \leq 8 \\ & x_1 + x_2 - x_3 \leq 4 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Suppose that we wanted to estimate how large the optimal solution's objective value was. We could get the exact answer by running the simplex algorithm, but we could get a quick upper-bound by examining the constraints. For example, the first constraint says that $x_1 + x_2 + x_3 \leq 10$. If we multiply both sides by 3, we get an equivalent inequality $3x_1 + 3x_2 + 3x_3 \leq 30$, which any feasible solution to our program must satisfy. Then, since $x_1, x_2, x_3 \geq 0$, we have:

$$2x_1 + 3x_2 + x_3 \leq 3x_1 + 3x_2 + 3x_3 \leq 30\,, \tag{7.1}$$

and so we know that no feasible solution to our program has objective value more than 30. We can do better than this, though. Suppose we multiply the second inequality by 2 and add it to the first inequality. Then, we get:

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &\leq (x_1 + x_2 + x_3) + 2\left(\frac{1}{2}x_1 + x_2\right) \\ &\leq 10 + 2 \cdot 8 = 26\,. \end{aligned} \tag{7.2}$$

Alternatively, we could multiply the first inequality by 2 and add it to the second to get:

$$\begin{aligned} 2x_1 + 3x_2 + x_3 \leq 3x_1 + 3x_2 + x_3 &= 2(x_1 + x_2 + x_3) + (x_1 + x_2 - x_3) \\ &\leq 2 \cdot 10 + 4 = 24\,. \end{aligned} \tag{7.3}$$

In order to compute each of the bounds (7.1)–(7.3), we multiplied our constraints by some constants and added them together. Our goal was to do this in a way that made the right-hand side as small as possible. However, we needed to make sure that the coefficient of each x_i was *at least* as large as its coefficient c_i in the objective. This gave us the first inequality in each chain of inequalities above. Also, we needed to make sure that the numbers we multiplied by were non-negative, so that the inequalities in the constraints did not change direction. This allowed us to apply the second inequality in each of the examples above.

In order to make this formal, suppose that we want to find an upper bound on the objective for a linear program in standard inequality form:

$$\begin{aligned} \text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots c_nx_n \\ \text{subject to} \quad & a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n \leq b_1 \\ & a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n \leq b_2 \\ & \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ & a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n \leq b_m \\ & \qquad\qquad\qquad\qquad\qquad x_j \geq 0 \text{ for each } j = 1, \ldots, n \end{aligned} \tag{7.4}$$

Then, our problem becomes the following. We want to choose values $y_1, \ldots, y_m$ by which to multiply each constraint. Our goal is to make the total right-hand side, given by

$$b_1y_1 + b_2y_2 + \cdots + b_my_m\,,$$

as small as possible. The total coefficient of the variable x_j in our resulting left-hand side will be given by

$$a_{1,j}y_1 + a_{2,j}y_2 + \cdots + a_{m,j}y_m\,.$$

As we discussed before, for each x_j, we want to make sure that this is at least c_j. Also, we need to make sure that $y_i \geq 0$.

It turns out we can formulate all of this as another linear program:

$$\begin{aligned}
\text{minimize} \quad & b_1y_1 + b_2y_2 + \cdots b_my_m \\
\text{subject to} \quad & a_{1,1}y_1 + a_{2,1}y_2 + \cdots + a_{m,1}y_m \geq c_1 \\
& a_{1,2}y_1 + a_{2,2}y_2 + \cdots + a_{m,2}y_m \geq c_2 \\
& \qquad \vdots \qquad\qquad\qquad \vdots \\
& a_{1,n}y_1 + a_{2,n}y_2 + \cdots + a_{m,n}y_m \geq c_n \\
& \qquad\qquad\qquad\qquad y_i \geq 0 \text{ for each } i = 1, \ldots, m
\end{aligned} \tag{7.5}$$

We say that the program (7.5) is the *dual* of (7.4). In order to keep track of which we are talking about, it is customary to refer to our original program (here, (7.4)) as the *primal* program.

Notice that we can formulate both of these programs even more succinctly using matrix notation. If our primal program has the form:

$$\begin{aligned}
\text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0}
\end{aligned}$$

with n variables and m constraints, then our dual program will always have the form:

$$\begin{aligned}
\text{minimize} \quad & \mathbf{b}^\mathsf{T}\mathbf{y} \\
\text{subject to} \quad & A^\mathsf{T}\mathbf{y} \geq \mathbf{c} \\
& \mathbf{y} \geq \mathbf{0}
\end{aligned}$$

with m variables and n constraints. Notice that each constraint of the primal becomes a variable in the dual, and each variable of the primal becomes a constraint in the dual. Note that the dual of a maximisation program will always be a minimisation problem (and, it turns out, vice versa). It is relatively easy to see from the above formulation that the dual of the dual program is again the primal program. It follows that each primal has a unique dual, and that once you know either the primal or the dual program, you can always figure out the other one.

In the above, we considered a linear program in standard inequality form. In general, we will typically assume that one of our programs (usually the primal) has maximisation as the goal, all inequalities of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$, and all variables non-negative or unrestricted. Inequalities of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \geq b$ and variables that are non-positive can both easily be handled by changing signs in our program. However, as we have seen, handling equations and unrestricted variables is slightly

more tedious and increases the number of constraints and variables in our program. It ends up that we can deal with both of these directly.

Intuitively, if our primal program has an equation, we can go through the same sort of argument as above to find an upper bound on its objective. The key difference is that we can multiply an equation by *either a positive or negative value* without changing its meaning. Thus, when we construct the dual of a linear program, each *equation constraint* in the primal will become an *unrestricted variable* in the dual. The reverse is true as well: suppose a program has an unrestricted variable x_i. Then when we construct our bound on the program's objective, it's no longer enough to make sure that coefficient of x_i that we get when adding up the constraints is at least c_i. This is because x_i could be negative, and then our resulting inequality would be the wrong way around. Instead, we must make sure that the coefficient of x_i is exactly the same as c_i. That is, each *unrestricted variable* in the primal will become an *equation* in the dual.

The following table shows the general relationship between objects in the primal and the dual:

maximise	$\iff$	minimise
non-negative variable x_j	$\iff$	$\geq$ constraint
unrestricted variable x_j	$\iff$	$=$ constraint
objective $\mathbf{c}^\mathsf{T}\mathbf{x}$	$\iff$	RHS $\mathbf{c}$
LHS $A\mathbf{x}$	$\iff$	LHS $A^\mathsf{T}\mathbf{y}$
RHS $\mathbf{b}$	$\iff$	objective $\mathbf{b}^\mathsf{T}\mathbf{y}$
$\leq$ constraint	$\iff$	non-negative variable y_i
$=$ constraint	$\iff$	unrestricted variable y_i

Note that this table can be used in both directions, but we need to be careful that our program is in an appropriate form. Consider now a general maximise program with some inequalities, some equations, some non-negative variables, and some unrestricted variables. We form the dual of this program as follows:

1. Make sure all inequalities are of the form $\mathbf{a}^\mathsf{T}\mathbf{x} \leq b$ and that all variables are either unrestricted or non-negative. This can be done by multiplying ($\geq$) inequalities by -1 and introducing variables $\bar{x} = -x$ for each non-positive variable x.
2. Introduce a variable y_i for each constraint. If the corresponding constraint is an equation, then y_i will be unrestricted. If the constraint is a ($\leq$) constraint, then we will have $y_i \geq 0$.

3. Set the goal of the program to be minimise and construct the objective function $\mathbf{b}^\mathsf{T}\mathbf{y}$ by using the right-hand side (RHS) vector $\mathbf{b}$ for the primal constraints. That is, in the dual objective, the coefficient of a variable y_i is just the left-hand side of the ith constraint in the primal.
4. Construct a constraint in the dual corresponding to each primal variable x_i, as follows:
   (a) The left-hand side (LHS) of the constraint corresponding x_i has the form:
   $$a_{1,i}y_1 + a_{2,i}y_2 + \cdots + a_{m,i}y_m\,.$$
   That is the coefficients for the dual variables in the ith dual constraint are given by the ith column of the matrix A from the primal, which gives the coefficients of x_i in each of the m constraints in the primal.
   (b) The right-hand side (RHS) of the constraint corresponding to x_i is given by the coefficient c_i of x_i in the primal.
   (c) Finally, you need to decide if this constraint is an $=$ or $\geq$ constraint. If the primal variable x_i is unrestricted, then this corresponding dual constraint will be an $=$ constraint. If the primal variable has $x_i \geq 0$, then the corresponding dual constraint will be $\geq$.

If we want to take form the dual of a program of the form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{b}^\mathsf{T}\mathbf{y} \\ \text{subject to} \quad & A\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned}$$

we can proceed in almost the same fashion, except the correspondences in the table above will go from right to left. Considering the above program now as the primal, we get the following. The dual will be a maximisation program, with a variable x_i for each constraint of the above primal program. In the dual objective, the coefficient of x_i is given by the RHS of the corresponding, ith constraint of the primal (that is, by c_i). Each variable y_j of the primal program will correspond to a dual constraint that has LHS given by the jth column of A and has RHS given by the corresponding entry b_j. This constraint will be either $\leq$ or $=$ depending on whether $y_j \geq 0$ or y_j unrestricted, respectively. Finally in the dual's objective, x_i will have coefficient c_i, given by the RHS of the corresponding ith constraint in the primal program.

Example 7.1. Give the dual of the following linear program:

$$\begin{aligned}
\text{maximize} \quad & x_1 + 2x_2 + 3x_3 + 4x_4 \\
\text{subject to} \quad & x_1 - 3x_3 + 9x_4 \leq 10 \\
& 3x_1 + x_2 + 5x_3 + 7x_4 \leq 5 \\
& x_2 + 7x_4 \geq 13 \\
& 7x_1 + 11x_2 + 9x_3 - 7x_4 = 25 \\
& x_1 + x_2 + x_3 + x_4 = 100 \\
& x_1, x_4 \geq 0 \\
& x_2, x_3 \text{ unrestricted}
\end{aligned}$$

Solution. Here, we have $\mathbf{c}^\mathsf{T} = (1, 2, 3, 4)$ and also after making sure all constraints are in the right direction for maximisation (that is, either $\leq$ or $=$), we get:

$$A = \begin{pmatrix} 1 & 0 & -3 & 9 \\ 3 & 1 & 5 & 7 \\ 0 & -1 & 0 & -7 \\ 7 & 11 & 9 & -7 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 10 \\ 5 \\ -13 \\ 25 \\ 100 \end{pmatrix}$$

So, we will have 5 dual variables $y_1, \ldots, y_5$, one for each constraint of the primal (that is, each row of A). Looking at the primal constraints, we see that we will have $y_1, y_2, y_3 \geq 0$ and y_4, y_5 unrestricted, since the first 3 constraints are ($\leq$) constraints and the last 2 are ($=$) constraints.

Our dual objective function will be:

$$\mathbf{b}^\mathsf{T}\mathbf{y} = 10y_1 + 5y_2 - 13y_3 + 25y_4 + 100y_5$$

Our constraints will have left-hand side $A^\mathsf{T}\mathbf{y}$ and right-hand side $\mathbf{c}$. The first and fourth constraints (corresponding to the columns from A for x_1 and x_4) will be $\geq$ and the second and third (corresponding to the columns from A for x_2 and x_3) will be $=$. Altogether, we get:

$$\begin{aligned}
\text{minimize} \quad & 10y_1 + 5y_2 - 13y_3 + 25y_4 + 100y_5 \\
\text{subject to} \quad & y_1 + 3y_2 + 7y_4 + y_5 \geq 1 \\
& y_2 - y_3 + 11y_4 + y_5 = 2 \\
& -3y_1 + 5y_2 + 9y_4 + y_5 = 3 \\
& 9y_1 + 7y_2 - 7y_3 - 7y_4 + y_5 \geq 4 \\
& y_1, y_2, y_3 \geq 0 \\
& y_4, y_5 \text{ unrestricted}
\end{aligned}$$

□

7.1 The Weak and Strong Duality Theorems

Henceforth, we focus on primal linear programs in standard inequality form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{7.6}$$

Thus, our dual program will always have the form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{b}^\mathsf{T}\mathbf{y} \\ \text{subject to} \quad & A^\mathsf{T}\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{7.7}$$

For our dual program, we have that a feasible solution $\mathbf{y}^*$ is optimal if and only if $\mathbf{b}^\mathsf{T}\mathbf{y}^* \leq \mathbf{b}^\mathsf{T}\mathbf{y}$ for every other feasible solution $\mathbf{y}$. Note that the inequality is reversed—since we are trying to *minimise*, the best solution should have objective value as small as possible.

We introduced the dual of a linear program as a way to place an upper bound on its optimal value. The following theorem formalises this idea, by showing that every feasible solution to a program's dual gives us an upper bound on the value of any of this program's feasible solutions.

Theorem 7.1 (Weak Duality Theorem for Linear Programs)**.** Consider a linear program in standard inequality form. Let $\mathbf{x}$ be any feasible solution to this program and let $\mathbf{y}$ be any feasible solution to the dual of this program. Then:

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

That is, the objective value of $\mathbf{x}$ in the original program is at most the objective value of $\mathbf{y}$ for the dual.

Proof. Consider a feasible solutions $\mathbf{x}$ of a program in the form (7.6), and $\mathbf{y}$ of the associated dual in the form (7.7). Since $\mathbf{y}$ is feasible for the dual, we must have:

$$c_j \leq \sum_{i=1}^{m} a_{i,j} y_i,$$

for each $j = 1, \ldots, n$. Since $\mathbf{x}$ is feasible for the primal, we must have $x_j \geq 0$ and so multiplying both sides of the above inequality by x_j gives:

$$c_j x_j \leq \sum_{i=1}^{m} a_{i,j} x_j y_i \tag{7.8}$$

Similarly, since $\mathbf{x}$ is feasible for the primal, we must have: s

$$\sum_{j=1}^{n} a_{i,j} x_j \leq b_i$$

for each $i = 1, \ldots, m$. Since $\mathbf{y}$ is feasible for the dual, we must have $y_i \geq 0$ and so multiplying both sides of the above inequality by y_i gives:

$$\sum_{j=1}^{n} a_{i,j} x_j y_i \leq b_i y_i \tag{7.9}$$

The rest is easy. We have:

$$\mathbf{c}^\mathsf{T}\mathbf{x} = \sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} \sum_{i=1}^{m} a_{i,j} x_j y_i = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j} x_j y_i \leq \sum_{i=1}^{m} b_i y_i = \mathbf{b}^\mathsf{T}\mathbf{y}$$

where the first inequality follows from applying (7.8) to each term of the summation, and the second follows from applying (7.9) to each term. □

Weak duality says that the objective value of every feasible solution for a dual program gives us an upper bound on the objective value of any feasible solution for its associated primal program. In fact, we can say something much stronger, which is that at an optimal solution, the dual and primal objectives are in fact equal. This is called the Strong Duality Theorem and will be covered next week.

Week 8

Duality (II)

Last week we proved the Weak Duality Theorem. Weak duality says that the objective value of every feasible solution for a dual program gives us an upper bound on the objective value of any feasible solution for its associated primal program. In fact, we can say something much stronger, which is that at an optimal solution, the dual and primal objectives are in fact equal. This is called the Strong Duality Theorem and will be covered next week.

Theorem 8.1 (Strong Duality Theorem for Linear Programs)**.** If a linear program (in standard inequality form) has an optimal solution $\mathbf{x}^*$ with objective value z^*, then there is an optimal solution $\mathbf{y}^*$ to its dual with objective value z^*.

Proof. Consider an optimal solution $\mathbf{x}^*$ of the primal program. We will show how to construct a feasible solution $\mathbf{y}^*$ of the dual program whose dual objective value satisfies $\mathbf{b}^\mathsf{T}\mathbf{y}^* = \mathbf{c}^\mathsf{T}\mathbf{x}^*$. By weak duality we then must have that $\mathbf{y}^*$ is optimal, since every other feasible solution $\mathbf{y}$ of the dual must have $\mathbf{y}^\mathsf{T}\mathbf{b} \geq \mathbf{x}^{*\mathsf{T}}\mathbf{c} = \mathbf{y}^{*\mathsf{T}}\mathbf{b}$.

Since the primal program has an optimal solution, we know that applying the simplex algorithm to it will eventually give us a final tableau, in which the last row has no positive entries. Suppose the $-z$ row in our final tableau looks like:

$-z$	p_1	$\ldots$	p_n	q_1	$\ldots$	q_m	$-z^*$

(8.1)

where the entries $p_1, \ldots, p_n$ correspond to our original (primal) linear program's variables, the entries $q_1, \ldots, q_m$ correspond to the slack variables we introduced to bring the program into standard equation form, and $z^* = \mathbf{c}^\mathsf{T}\mathbf{x}^*$ represents the value of the optimal solution $\mathbf{x}^*$ corresponding to this tableau. Note that since this is the final tableau, we must have $p_j \leq 0$ and $q_i \leq 0$ for all $j = 1, \ldots, n$ and

$i = 1, \ldots, m$. Initially, our tableau was simply[1]:

	x_1	$\cdots$	x_n	s_1	$\cdots$	s_m	
s_1	$a_{1,1}$	$\cdots$	$a_{1,n}$	1	$\cdots$	0	b_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
s_m	$a_{m,1}$	$\cdots$	$a_{m,n}$	0	$\cdots$	1	b_m
$-z$	c_1	$\cdots$	c_n	0	$\cdots$	0	0

Each pivot step of the simplex algorithm adds some multiple of one row to all of the other rows and to the objective function. Thus, at the end, our final row will be our initial row plus some constant times each initial row. How can we find out exactly how much of each initial row got added to the objective row throughout the algorithm? Initially this seems quite hopeless—for example, our first pivot will add some amount of the pivot row to all of the other rows (including the last row). In later steps, when one of the other rows is chosen as the pivot row, it will therefore be equal to its initial value *plus some amounts of all of our previous pivot rows*. Luckily, there is an easy way to figure out. As an example, let's look at the initial and final tableaux for our first example problem from Section 5.2. If we label our columns so that we can identify our original variables and our slack variables, we find the following initial and final tableaux:

	x_1	x_2	s_1	s_2	s_3	
s_1	1	1	1	0	0	3
s_2	$\frac{1}{2}$	1	0	1	0	2
s_3	$\frac{1}{2}$	-1	0	0	1	1
$-z$	4	$\frac{1}{2}$	0	0	0	0

	x_1	x_2	s_1	s_2	s_3	
x_2	0	1	$\frac{1}{3}$	0	$-\frac{2}{3}$	$\frac{1}{3}$
s_1	0	0	$-\frac{2}{3}$	1	$\frac{1}{3}$	$\frac{1}{3}$
x_1	1	0	$\frac{2}{3}$	0	$\frac{2}{3}$	$\frac{8}{3}$
$-z$	0	0	$-\frac{17}{6}$	0	$-\frac{7}{3}$	$-\frac{65}{6}$

Notice that each slack variable has an entry of 1 in exactly one row, and zero in all of the others (including the row for $-z$) and that each row has exactly 1 slack variable with an entry 1. So, if we want to know the total amount of (for

[1]This is not strictly true for the 2-phase simplex method, but it ends up that the proof work there, too. We'll discuss this later. For now, let's just focus on the standard simplex algorithm.

example) the second row in the initial tableau that was added to the $-z$ row, over the entire algorithm, we can just look at the final entry in the column for its slack variable! For example, you can check that the second row in the final tableau is $-2/3$ times the first row of the initial tableau, plus 1 times the second row of the initial tableau, plus $1/3$ times the third row of the initial tableau. Thus, at any time during the algorithm, the slack variables will tell us what combination of the initial tableau's rows a given row represents. Similarly, the final value of the $-z$ row, will be its initial value[2] plus some combination of the rows from the initial tableau. We can find this combination by looking at the slack variables. Here, we see that the $-z$ row of the final tableau should then be equal to its initial value, plus $-\frac{17}{6}$ times the first row of the initial tableau, plus $-\frac{7}{3}$ times the third row of the initial tableau. You can check that this is in fact the case, here.

We can use this trick in general. The $-z$ row in our final tableau will be equal to the $-z$ row in the initial tableau plus q_1 times the 1st row, q_2 times the second, and so on (where the q_i are the entries for the slacks of the $-z$ row, as shown in (8.1)). Thus, for each $j = 1, \ldots, n$, the jth entry in our final row must be:

$$p_j = c_j + \sum_{i=1}^{m} q_i a_{ij} \tag{8.2}$$

and, similarly the last entry must be given by:

$$-z^* = 0 + \sum_{i=1}^{m} q_i b_i \tag{8.3}$$

Now, let's define our dual solution by setting $y_i^* = -q_i$. Note that this makes $y_i^* \geq 0$, as required, since each $q_i \leq 0$. Moreover, (8.2) is equivalent to:

$$c_j = p_j - \sum_{i=1}^{m} q_i a_{ij} = p_j + \sum_{i=1}^{m} y_i^* b_i \leq \sum_{i=1}^{m} y_i^* b_i$$

for each $j = 1, \ldots, n$. The first equation is obtained by rewriting (8.2), the second is just from our definition $y_i^* = -q_i$, and the last inequality follows from the fact that $p_j \leq 0$ since it is an entry in the $-z$ row of the final tableau. Note that the inequalities we have obtained show exactly that $\mathbf{y}^*$ satisfies all constraints of the dual and so indeed $\mathbf{y}^{*\mathsf{T}} = (-q_1, -q_2, \ldots, -q_m)$ is a feasible solution to our

[2]Why do we need to explicitly remember this for the $-z$ row, but not the others? The answer is that the other rows had a slack variable initially set to 1, which captured the fact that there was 1 times the initial value of this row in the resulting combination. However, the $-z$ row does not have its own slack variable, so we need to explicitly remember to add in the initial value.

dual program. To complete the proof, we just need to compute the value of this solution. It is exactly:

$$\sum_{i=1}^{m} y_i^* b_i = -\sum_{i=1}^{m} q_i b_i = -(-z^*) = z^* \,,$$

where we have used our definition of y^* for the first equation and (8.3) for the second.

Altogether, we have shown a general method that always finds a feasible dual solution with objective value equal to that of the optimal primal solution. □

In our proof we supposed that every row of the tableau had some slack variable. Since we assumed our program was in standard inequality form, every row will have a slack variable when we formulate the initial tableau. However, if the right-hand side of any of our constraints is negative, we still must use the 2-phase simplex algorithm to solve the problem. We briefly note that the above argument works in that case, too, with only small modifications. Specifically, we consider the tableau constructed at the end of step 4 of procedure (that is, the tableau we get before fixing signs and adding each artificial row to the $-w$ row). In this tableau, we have introduced a slack variable with coefficient 1 to each inequality, together with some artificial variables (for the rows with a negative right-hand side). Since we are assuming that our linear program has an optimal solution, the first phase must terminate with all artificial variables set to zero. We then carry over the $-z$ row to the tableau for the second phase. You can verify that at the end of the second phase, the row for $-z$ is indeed equal to its initial value at step 4 of the 2-phase algorithm plus some amount of the initial values of each row in step 4. Indeed, each pivot operation in either phase will add some amount of the pivot row to the $-z$ row just as in our above argument. The only difference is that these amounts will accumulate over both phases. Some of the rows will have multiplied by -1 when we fix the tableau after step 4, but this makes no difference to our argument: we don't care whether a given row is added or subtracted at each step, and so this sign change will already incorporated into the book-keeping for the total "amount" of a row that we are taking. Just as before, then, the total amount of an initial row that was added into the $-z$ row (over both phases) will be given by the value of that row's slack variable at the end of phase 2.

We have proved the weak and strong duality theorem for linear programs. One extremely useful consequence of the strong duality theorem is that we can now easily show that a given solution to a linear program is optimal.

If we want to show that a solution is feasible, things are easy: we can just plug in the values for our variables and check by hand that each of the program's equations and inequalities are true. On the other hand, to show that a solution

is optimal, we need to show that its objective value is at least as large as that of every other feasible solution. This is much more difficult, since we need to prove a statement about *all* feasible solutions. However, the strong duality theorem lets us check that a solution $\mathbf{x}$ is optimal for some primal linear program by simply verifying that *a single*, given solution $\mathbf{y}$ to its dual is feasible. If $\mathbf{x}$ and $\mathbf{y}$ have the same objective values in the primal and dual, respectively, we then know that both of them must be optimal.

As an example, suppose that you wanted to convince your supervisor that you had found the best possible solution to a production problem. You could of course say that you had found it by applying the simplex algorithm and so it must be optimal, but what if you made a mistake in some pivot operation? If you used a computer, how could you be sure that the code for simplex did everything correctly? For important problems, it is often necessary to double-check a solution by hand. You could simply show your supervisor a solution to the dual of your problem. It would then be easy for her to check that both solutions had the same objective value and that your dual solution was feasible. The strong duality theorem guarantees that you can always find some such dual solution to convince her. This dual solution is sometimes called a *certificate of optimality*. We call it a "certificate" because it is a succinct, easily verifiable proof that a solution is optimal. The strong duality theorem shows that every linear program that has an optimal solution also has a succinct certificate proving that this is optimal.

8.1 Complementary Slackness

How do we find a dual solution to serve as a certificate? Looking at the proof of the strong duality theorem, we see that we can easily obtain it from the last row of the tableau. What if you didn't have the entire simplex tableau? In this lecture, we will see another property that allows us to easily show that a given solution to a linear program is optimal.

Theorem 8.2 (Principle of Complementary Slackness)**.** Suppose that $\mathbf{x}$ is a feasible solution to a (primal) linear program in standard inequality form, and let $\mathbf{y}$ be a feasible solution to its dual.

Then, $\mathbf{x}$ and $\mathbf{y}$ are optimal if and only if both of the following hold:

- For every $j = 1, \ldots, n$, $\quad x_j = 0 \quad$ or $\quad \sum_{i=1}^{m} a_{i,j} y_i = c_j$.
- For every $i = 1, \ldots, m$, $\quad y_i = 0 \quad$ or $\quad \sum_{j=1}^{n} a_{i,j} x_j = b_i$.

Intuitively complementary slackness says that a pair of feasible solutions $\mathbf{x}$ and $\mathbf{y}$ to a linear program and its dual are both optimal if and only if for every primal variable x_j that is not zero, the corresponding constraint of the dual is tight, and for every variable y_i that is not zero, the corresponding constraint of the primal is tight. We now prove that this is the case.

Proof of the Principle Complementary Slackness for Linear Programs. The proof follows by considering the inequalities we used in the proof of the weak duality theorem. Let $\mathbf{x}$ and $\mathbf{y}$ be optimal basic feasible solutions to a pair of primal and dual linear programs, respectively. Then, as discussed in the proof of weak duality for linear programs, primal and dual feasibility imply the following inequalities:

$$c_j x_j \leq \sum_{i=1}^{m} a_{i,j} x_j y_i \qquad \text{for all } j = 1, \ldots, n \tag{8.4}$$

$$\sum_{j=1}^{n} a_{i,j} x_j y_i \leq b_i y_i \qquad \text{for all } i = 1, \ldots, m \tag{8.5}$$

In that proof we summed the first inequality over all values of j and the second over all values of i, then combined them to get:

$$\mathbf{c}^\mathsf{T}\mathbf{x} = \sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} \sum_{i=1}^{m} a_{i,j} x_j y_i = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j} x_j y_i \leq \sum_{i=1}^{m} b_i y_i = \mathbf{b}^\mathsf{T}\mathbf{y}$$

Now, notice that if any of our inequalities of the form (8.4) and (8.5) is not tight (that is, if it holds with $<$), then we will have $\mathbf{c}^\mathsf{T}\mathbf{x} < \mathbf{b}^\mathsf{T}\mathbf{y}$ above. On the other hand, if every inequality of the form (8.4) and (8.5) is tight (that is, if all of them hold with equality), then both of the inequalities above will also be tight, and so we will actually have $\mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{b}^\mathsf{T}\mathbf{y}$.

It follows that $\mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{b}^\mathsf{T}\mathbf{y}$ (and so $\mathbf{x}$ and $\mathbf{y}$ are optimal) if and only if every inequality of the form (8.4) and (8.5) is tight. Rearranging both sets of inequalities, we see that this is equivalent to

$$x_j \left(c_j - \sum_{i=1}^{m} a_{i,j} y_i \right) = 0 \qquad \text{for all } j = 1, \ldots, n \tag{8.6}$$

$$y_i \left(b_i - \sum_{j=1}^{n} a_{i,j} x_j \right) = 0 \qquad \text{for all } i = 1, \ldots, m \tag{8.7}$$

Now, we simply observe that (8.6) holds if and only if $x_j = 0$ or $\sum_{i=1} a_{i,j} y_i = c_j$ for every $j = 1, \ldots, n$ and (8.7) holds if and only if $y_i = 0$ or $\sum_{j=1}^{n} a_{i,j} x_j = b_i$ fore all $i = 1, \ldots, m$. Thus, we have shown that $\mathbf{x}$ and $\mathbf{y}$ are optimal if and only if the complementary slackness conditions hold. □

Using complementary slackness, we can derive an optimal feasible dual solution from an optimal basic feasible solution $\mathbf{x}$ to a primal linear program. That is, we can easily find our desired certificate of optimality. We do this as follows:

- First, we look at the values of the primal variables x_j. If $x_j \neq 0$, then we know that the jth constraint of the dual linear program must be tight.
- Next, we look at the primal constraints, and plug in the values of our variables from $\mathbf{x}$. If these values do not make the ith primal constraint tight, then the ith dual variable y_i must be 0.
- Altogether, we get a system of equations for our dual variables, which we can solve to get values for $\mathbf{y}$. If we can find one such solution $\mathbf{y}$ that satisfies all of the dual constraints (that is, if our $\mathbf{y}$ is feasible) then $\mathbf{x}$ and $\mathbf{y}$ are both optimal. If we can show that there is no such solution, then $\mathbf{x}$ is not optimal.

Let's see an example of how this works.

Example 8.1. Consider the linear program:

$$\begin{aligned} \text{maximize} \quad & 2x_1 - x_2 + 8x_3 \\ \text{subject to} \quad & 2x_3 \leq 1 \\ & 2x_1 - 4x_2 + 6x_3 \leq 3 \\ & -x_1 + 3x_2 + 4x_3 \leq 2 \\ & x_1, x_2, x_3 \geq 0 \end{aligned} \tag{8.8}$$

Show that $x_1 = 17/2$, $x_2 = 7/2$, $x_3 = 0$ is an optimal solution to this program.

Solution. It will be useful (but not strictly necessary) to rewrite our program to standard equation form. We get:

$$\begin{aligned} \text{maximize} \quad & 2x_1 - x_2 + 8x_3 \\ \text{subject to} \quad & 2x_3 + s_1 = 1 \\ & 2x_1 - 4x_2 + 6x_3 + s_2 = 3 \\ & -x_1 + 3x_2 + 4x_3 + s_3 = 2 \\ & x_1, x_2, x_3, s_1, s_2, s_3 \geq 0 \end{aligned} \tag{8.9}$$

Now, we can plug in the values for $\mathbf{x}$ that we were given, and find what values the slack variables must take in the program above. We find that when x_1, x_2, x_3 are set as in the example, we must have $s_1 = 1$, $s_2 = 0$, $s_3 = 0$. This tells us, first of all, that the given $\mathbf{x}$ is feasible—indeed, since all of the slack variables for (8.9) are non-negative, we satisfy all of the inequality constraints of our original program (8.8). It also lets us see right away which constraints of (8.8) are tight and which are not—the first constraint is *not* tight, since $s_1 > 0$, but the second and third constraints *are* tight, since $s_2 = s_3 = 0$. Now, we can apply the principle of complementary slackness to show that $\mathbf{x}$ is an optimal solution for (8.8).

First, we note that x_1 and x_2 are both non-zero, so our dual constraints corresponding to x_1 and x_2 must be tight. That is, we must have:

$$\begin{aligned} 2y_2 - y_3 &= 2 \\ -4y_2 + 3y_3 &= -1. \end{aligned}$$

Next, since the first constraint of (8.8) is not tight, its corresponding dual variable must be equal to 0. That is, we must have:

$$y_1 = 0.$$

Altogether, we get the following system of equations:

$$\begin{aligned} 2y_2 - y_3 &= 2 \\ -4y_2 + 3y_3 &= -1 \\ y_1 &= 0 \end{aligned}$$

This has a unique solution setting $y_1 = 0$, $y_2 = \frac{5}{2}$, $y_3 = 3$. Now, we only need to check that this is a feasible solution of our program's dual. First, we check that all the y variables are non-negative. Next, we check that our constraints are satisfied. We already know that the first and second constraints are satisfied, because we chose our values to satisfy the equations corresponding to these constraints being tight. The only remaining constraint to check is:

$$2y_1 + 6y_2 + 4y_3 \geq 8$$

Since $2 \cdot 0 + 6 \cdot \frac{5}{2} + 4 \cdot 3 = 27 \geq 8$, this constraint holds. Thus, our solution $\mathbf{y}$ is indeed a feasible solution to the dual. We constructed $\mathbf{y}$ to make sure that $\mathbf{x}$ and $\mathbf{y}$ satisfied the complementary slackness conditions, so it follows that $\mathbf{x}$ and $\mathbf{y}$ are optimal for our primal and dual programs, respectively. □

This same example program was solved with the simplex algorithm in Example 5.1. Notice that, as we discussed in our proof of strong duality, the values y_1,y_2, and y_3 correspond exactly to what we get if we negate the entries for our slack variables in the final simplex tableau there. However, here we were able to derive these just by looking at the program and the primal solution.

The above method will always give us a system of equations for the dual variables y_i. Here, we got 3 equations for 3 variables and found a unique solution. You may be wondering whether this is always the case. The answer is "yes," whenever the solution $\mathbf{x}$ is a non-degenerate basic feasible solution of the related program in standard equation form.

Formally, suppose we take a program in standard inequality form with n variables and m constraints and (as usual) introduce m slack variables to produce an equivalent program in standard equation form. For any setting of the original program's variables $x_1, \ldots, x_n$, the value of each slack variable is uniquely determined: it is equal to the right-hand side its corresponding inequality constraint minus the left-hand side. We can thus take the solution of our standard inequality form linear program and get a corresponding, unique solution to the equivalent standard equation form program. If $\mathbf{x}$ is a non-degenerate basic feasible solution, then it will have exactly m non-zero entries, and these will correspond to a linearly independent set of columns from A. These m non-zero variables will give us m linearly independent equations as follows: for each of our original variables x_j that is non-zero, complementary slackness gives us an equation $\mathbf{a}^\mathsf{T}\mathbf{y} = c$ where $\mathbf{a}$ is given by the column of the constraint matrix corresponding to x_j. For each slack variable s_i that is non-zero, we know that a primal constraint is *not* tight and so complementary slackness gives us an equation $y_i = 0$. Notice that this can be written as $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$, where $\mathbf{a}$ is the column of the constraint matrix corresponding to the slack variable s_i (recall that this column will be zero everywhere except in the ith row, where it is 1). Thus, we obtain m equations, and the left-hand side of each of these is given by one of the columns of A where $\mathbf{x}$ was non-zero. Since these columns are linearly independent, we have m linearly independent equations in our m variables y_i. It follows that these equations will always have a unique solution.

In the general case, applying the principle of complementary slackness to some solution $\mathbf{x}$ may give a system of equations for $\mathbf{y}$ with multiple solutions. In this case, we must simply find *some* solution $\mathbf{y}$ to this system that is feasible for the dual to show that $\mathbf{x}$ is optimal. In order to show that $\mathbf{x}$ is not optimal, we must

find a way to show that *no* solution of the complementary slackness equations is feasible for the dual.[3]

Finally, we note that complementary slackness can be applied even to programs with equations and unrestricted variables. By definition, an equation has no slack. Thus, if we have an equation in the primal it will not affect our general procedure, other than that we must remember to check that it is satisfied in when checking feasibility of $\mathbf{x}$ and remember that this equation's corresponding dual variable in $\mathbf{y}$ is unrestricted, and so is allowed take negative values when we check that $\mathbf{y}$ is feasible. Similarly, if we have an unrestricted variable in the primal, we must remember that it can take any value when checking that $\mathbf{x}$ is feasible, and we must ensure that its corresponding dual equation in the dual is satisfied when constructing $\mathbf{y}$.

8.2 Interpretation of the Dual

In many settings, the dual of a linear program may have some intuitive meaning or relationship to the problem being modelled by the primal program. In this section, we briefly discuss a few examples and see how an optimal solution to the dual may be useful for making decisions.

We have already seen one example of this in action. Our initial example of a linear program (Example 1.1) was a "blending problem" in which a student was trying to select a cheap blend of foods that met basic dietary requirements. In Example 1.2 we considered the same problem from the point of view of a vendor of dietary supplements trying maximise revenue. It should now be easy to see that this is in fact the dual of the program from Example 1.1, and so both problems have the same optimal objective value.

Now that we have developed a more comprehensive theory of linear programming, we know right away that, since the diet problem has 4 constraints (one for each of the 4 nutrients we consider) there must be an optimal solution that uses at most 4 foods. This is because we know that if this program is feasible and bounded, there must be an optimal solution that is a basic feasible solution. This solution will have at most $m = 4$ non-zero variables. In fact, we saw that it used only 3 foods—this is because one of the slack variables must have also been basic. The associated inequality is not tight (unless we are at a degenerate solution) and so we would thus expect that the dual would have its variable corresponding to this inequality set to zero. Indeed, if we compute an optimal dual solution using complementary slackness, we see that it sets: $y_1 = 1.43382$, $y_2 = 0.588235$,

[3] Alternatively, we could argue that no feasible solution to the dual satisfies the complementary slackness conditions for $\mathbf{x}$. In some situations, this approach might be preferable.

$y_3 = 0.0367647$, and $y_4 = 0$. These correspond to the nutrients Thiamin, Riboflavin, Niacin, and Vitamin C, respectively. By complementary slackness, this implies that the constraint for Vitamin C must not be tight (that is, the given diet gives more Vitamin C than is needed).

Suppose that (against her doctor's advice!) the student decided to consume less than the daily allowance of some nutrient. The dual variables tell us that consuming less Vitamin C won't make any difference, since this constraint is not even tight. However, consuming less of each of the other nutrients will lower the cost of the diet at the rate given by the dual variables. This makes sense from the point of view of the vitamin company—if the student consumes slightly less Thiamin, their revenue will decrease at a rate equal to the current price of Thiamin pills (which is y_1). Surprisingly, this is also true from the student's point of view. If she decides to consume slightly less Thiamin, then this will lower the value of the cheapest diet at a rate of 1.43382 (until perhaps some other constraint becomes tight, of course). The dual variables can thus be interpreted as telling our student how much satisfying each nutritional requirement is costing her.

Let's now see an example of duality in a production problem. Our first production problem (Example 2.1) involved a foundry making 2 kinds of parts with 4 different processes and constraints on metal, electricity and labour. We got the following linear program for maximising revenue.[4]

$$\begin{aligned}
\text{maximize} \quad & 4000x_1 + 1800x_2 + 4800x_3 + 11400x_4 \\
\text{subject to} \quad & 100x_1 + 70x_2 + 120x_3 + 270x_4 \leq 6000 \\
& 800x_1 + 600x_2 + 2000x_3 + 4000x_4 \leq 100000 \\
& 16x_1 + 16x_2 + 50x_3 + 48x_4 \leq 1000 \\
& x_i \geq 0, \quad \text{for each } i = 1, 2, 3, 4
\end{aligned}$$

The dual of this linear program is the following:

$$\begin{aligned}
\text{minimize} \quad & 6000y_1 + 100000y_2 + 1000y_3 \\
\text{subject to} \quad & 100y_1 + 800y_2 + 16y_3 \geq 4000 \\
& 70y_1 + 600y_2 + 16y_3 \geq 1800 \\
& 120y_1 + 2000y_2 + 50y_3 \geq 4800 \\
& 270y_1 + 4000y_2 + 48y_3 \geq 11400 \\
& y_i \geq 0, \quad \text{for each } i = 1, 2, 3
\end{aligned}$$

[4]Here, we consider the first program we derived, which does not involve input and output variables. We would get exactly the same result if we used the more complicated (but also more readable) program with i_1, i_2, i_3 and o_1, o_2. However, the dual would have some extra variables and would need to be simplified to see clearly what is going on.

We now have 1 constraint for each production process, and one variable representing the cost allocated to each requirement: y_1 represents metal, y_2 represents electricity, y_3 represents labour. One way of viewing the dual is as follows: we want to know what the total value of our current resources/assets. We have 6000 kg of metal, 100000 kWh of power, and 1000 hours of labour. Using these resources, we produce parts and obtain revenue. The dual variables represent how much one unit of each resource currently contributes to our overall revenue in the optimal primal solution. For example, the variable y_1 represents the rate at which our optimal total revenue would increase if we purchased a very small amount of additional metal—note that if we purchased a *large* amount of metal, we might end up with a completely different optimal production plan, but if we buy a sufficiently small amount, we can suppose that the optimal balance between production lines stays the same and we can then just use up this metal to make a few more of each part.

The variables of the dual then have an intuitive meaning as prices or values. Suppose we wanted to make a small change to our budget for next month. Should we increase the amount of metal on hand, or the amount of power or labour? What change would have the greatest immediate effect on revenue? The answer can be obtained by simply examining the dual variables. At an optimal solution, the dual variables for this particular program are $y_1 = 20$, $y_2 = 0$, $y_3 = 125$. We can get these either by looking at the final simplex tableau, or (if we only know the optimal solution to the primal) by using complementary slackness. Looking at the values of the dual variables, we see that currently we can increase our revenue the most by hiring labour, and that right now, this will increase our revenue by £125 per hour of labour hired. Again, note that this estimate is only valid for the current value of the linear program. That is, similarly to a derivative, it estimates the *instantaneous rate* of increase in revenue at the current solution point. If we hired 1000 hours of extra labour it might be that some other constraint would become tight and we would not see a full increase of $1000 \cdot 125$ in our revenue. Nonetheless, for very small increases, this is a good estimate just like the derivative gives a good estimate of the behaviour of a function for small changes around a single point.

By thinking about the dual in this way, we see that complementary slackness also has an intuitive meaning. If some constraint is *not* tight in the original program—for example the constraint on how much electricity we can use—then we would expect the corresponding dual variable—here, the value y_2 that tells us how much we can improve revenue by buying more electricity—to be 0. Indeed, buying more of any such will not help us to increase our revenue, since we are not even using all of what we already have! Note that this is exactly the principle of complementary slackness: wherever we assign a non-zero price to a resource in the dual, its corresponding constraint in the primal must be tight.

As the previous examples illustrate, there is often some meaning behind the value of the dual to a linear program. In general, each dual variable represents how much its constraint is affecting the solution. Its value tells us what rate the objective function would change at if we relaxed this constraint slightly. Thus, the dual variables can be extremely useful for making decisions about how to modify our plans.

8.3 Advanced Modelling: Handling Min-Max Problems

Before continuing on to game theory, let's return briefly to the question of what we can model with linear programs. We will see that there are a few "tricks" that allow certain kinds of non-linear objectives to be handled with a linear program. Both tricks rely on the fact that we have an algorithm for computing the optimal solution (namely, the simplex algorithm). Specifically, we will see that even though our linear programs in the next two sections may have feasible solutions that do not correspond to the reality of the problem we are modelling, we can guarantee that this will not happen for an *optimal* solution.

For our first problem, refer back to the following example from Week 2:

Example 8.2. A factory makes 2 different parts (say, part X and part Y). Their plant has 4 separate processes in place: there are two older processes (say, process 1 and 2) that produce parts X and Y directly, as well as two different integrated processes for producing both X and Y simultaneously. The 4 processes can be run simultaneously, but require labour, raw metal, and electricity. The hourly inputs and outputs for each process are as follows:

	Outputs		Inputs		
Process	X	Y	Metal	Electricity	Labour
1	4	0	100 kg	800 kWh	16 hrs
2	0	1	70 kg	600 kWh	16 hrs
3	3	1	120 kg	2000 kWh	50 hrs
4	6	3	270 kg	4000 kWh	48 hrs

Suppose now that we wanted to know *how fast* we could produce 120 of part X and 50 of part Y, if we had *unlimited resources*. This could be used to find out, for example, the absolute minimum number of hours we need to keep the factory open to still meet our production requirements. To figure this out we need to see which of our processes runs for the *longest time*. The quantity we are interested

in is thus given by the maximum of x_1, x_2, x_3, x_4. But, is this a linear function? No! For example $\max(2, 0, 0, 0) = 2$ and $\max(0, 1, 0, 0) = 1$ but:

$$\max(2 + 0, 0 + 1, 0 + 0, 0 + 0) = 2 \neq 2 + 1\,.$$

But, what do we know about the maximum of a set? Obviously, it must be larger than any element of the set. Recall that we introduced decision variables p_1, p_2, p_3, p_4 representing how long to run each process when modelling this problem in Week 2. Let's introduce a new variable m to represent the maximum of p_1, p_2, p_3, p_4. Then, we know that:

$$\begin{aligned} m &\geq p_1 \\ m &\geq p_2 \\ m &\geq p_3 \\ m &\geq p_4 \end{aligned}$$

These are *definitely* linear constraints. Let's add m to our production program, together with these constraints, and then just try to minimise m. Doing this we get something like the following:

$$\begin{aligned} \text{minimize} \quad & m \\ \text{subject to} \quad & m \geq p_j \qquad \text{for each } j = 1, 2, 3, 4 \\ & x = 4p_1 + 3p_3 + 6p_4 \\ & y = p_2 + p_3 + 3p_4 \\ & x \geq 120 \\ & y \geq 50 \\ & m \geq p_1 \\ & m \geq p_2 \\ & m \geq p_3 \\ & m \geq p_4 \\ & p_1, p_2, p_3, p_4 \geq 0 \\ & x, y, m \text{ unrestricted} \end{aligned} \tag{8.10}$$

Here, the first 2 constraints tell us how many parts X and Y we can make when we run our processes for p_1, p_2, p_3, p_4 hours, respectively. The second 2 constraints say that we must produce at least the required number of parts X and Y. The last 4 constraints say that the variable m representing the maximum of p_1, p_2, p_3, p_4 must be at least as large as each of p_1, p_2, p_3, p_4. Notice that, in general, the feasible solutions of this program will not correspond exactly to what we want, since the

maximum is *equal* to the largest value of p_1, p_2, p_3, p_4, but here we only ask that m be *at least* each of these values. However, we will show that for an *optimal* solution to our program, then m will always be equal to the maximum of p_1, p_2, p_3, p_4. This is good enough for us, assuming our goal is to actually solve the problem optimally.

Intuitively, the idea is that as long as m is strict larger than the maximum of $p_1, \ldots, p_4$, we could decrease it to get a better solution for our program. Formally, consider any feasible solution to our program, and suppose that it assigns values $p_1^*, \ldots, p_4^*$ and m^* to the variables $p_1, \ldots, p_4$ and m. Then, since the solution is feasible, we must have that $p_i^* \leq m^*$ for each $i = 1, 2, 3, 4$. Suppose that $m^* > \max(p_1^*, \ldots, p_4^*)$ and set $\varepsilon = m^* - \max(p_1^*, \ldots, p_4^*) > 0$. Then, $m^* - \varepsilon = \max(p_1^*, \ldots, p_4^*) \geq p_i^*$ for each $i = 1, 2, 3, 4$, so setting $m = m^* - \varepsilon$ gives a feasible solution to the linear program—note that we only changed m and it does not occur in any other constraints, so they are still satisfied. This feasible solution also has an objective value that is ε *smaller* than our original Thus, our original solution *could not have been optimal*. It follows any optimal solution to our transformed program must have $m = \max(p_1, \ldots, p_4)$, just as we claimed.

Note that we can repeat the above procedure in any mathematical program that requires we *minimise* the *maximum* of some set of variables. Whenever we have a mathematical *minimisation* program with an objective of the form:

$$\max(x_1, \ldots, x_k)$$

for some set of variables (here we call them $x_1, \ldots, x_k$), we do the following:

1. We change the program by introducing a new unrestricted variable (let's call it m).
2. We replace the term $\max(x_1, \ldots, x_k)$ in the objective by m.
3. We add a constraint saying that m should be larger than each of the variables in the maximum. That is, for each variable x_i that appears in the maximum, we introduce the constraint:
$$m \geq x_i$$

The above procedure will give us a linear program, by getting rid of the maximum. We can use a similar trick to model any linear program that involves *maximising* the *minimum* of several variables. There, we replace $\min(x_1, \ldots, x_k)$ by m and add constraints $m \leq x_i$ for each x_i.

8.4 Piecewise Linear Concave and Convex Objectives

Let's look at another production example, which is a modification of Example 9.1.

Example 8.3. Consider the setting from Example 9.1 and suppose that, as before, each unit of part Y sells for £1800, but that due to market demand, we can only sell the first 30 units of part X for £1000. Further units up to 60 can be sold for £700 and any excess units after that can be sold only for £400. Suppose you have 6000 kg of metal, 100000 kWh of electricity, and 1000 h of labour available. How should we schedule production to maximise revenue?

Here, the revenue from selling X is given by the function f depicted in Figure 9.1. Note that this is *not* a linear function. It is a combination of several different linear functions "glued together." We call such functions *piecewise linear*. Note that the function f behaves like a linear function f_1 passing through the origin with slope 1000 when $x \in [0, 30)$, like a linear function f_2 with slope 700 when $x \in [30, 60)$, and like a linear function f_3 with slope 400 when $x \in [60, \infty)$. We say that the function f is *concave* because these slopes are *decreasing*. This should agree roughly with your intuition for what graphs of concave functions look like.

We cannot model this problem directly as a linear program, since our profit function is non-linear. However, we will see that since it is *piecewise* linear and *concave* we can still *maximise* it using a linear program. To do this, suppose that x is the decision variable for how many of part X we make—that is, x is the argument of our piecewise linear concave function f. We want to find an expression for each linear piece of the function f.

Let's start with the first piece f_1. We know that the slope of f_1 is 1000 and f_1 passes through the origin, so $f_1(0) = 0$. Thus, we have:

$$f_1(x) = 1000x + 0 = 1000x$$

Now, let's look at f_2. We know that the slope of f_2 is 700, so f_2 has the general form:

$$f_2(x) = 700x + b_2$$

for some constant b that we will now determine. Intuitively, b determines how much we "shift" the for f_2 up from the origin, and we want to shift it so that it joins up with the line for f_1 at the point $x = 30$, where f stops behaving like

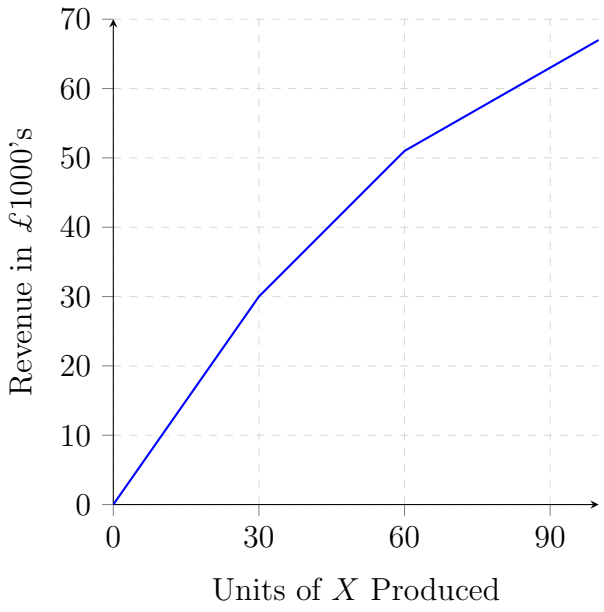


Figure 8.1: Revenue for Example 9.2

f_1 and starts behaving like f_2. Formally, since f is continuous[5], we must have $f_1(30) = \lim_{x\to 30^+} f(x) = \lim_{x\to 30^-} f_2(30) = f_2(30)$. Using the expression we already have for $f_1(x)$ we get:

$$f_1(30) = 1000 \cdot 30 \qquad\qquad f_2(30) = 700 \cdot 30 + b_2$$

and so:

$$1000 \cdot 30 = 700 \cdot 30 + b_2$$

which we can solve to get $b_2 = 9000$. Thus:

$$f_2(x) = 700x + 9000.$$

We can now repeat the above procedure to find an equation for f_3. Remember that $f(x)$ behaves like $f_2(x)$ for $x \in [30, 60)$ and like $f_3(x)$ for $x \in [60, \infty)$. So, now we need $f_2(60) = f_3(60)$, in order for these 2 "pieces" to join up at 60. We know the slope of f_3 is 400 and we need to find its intercept b_3. We now have:

$$f_2(60) = 700 \cdot 30 + 9000 \qquad\qquad f_3(60) = 400 \cdot 60 + b_3$$

[5]If you've not encountered the formal definition of continuity: all we are saying here is that the "piece" for f_1 must end at the same value that the "piece" for f_2 starts.

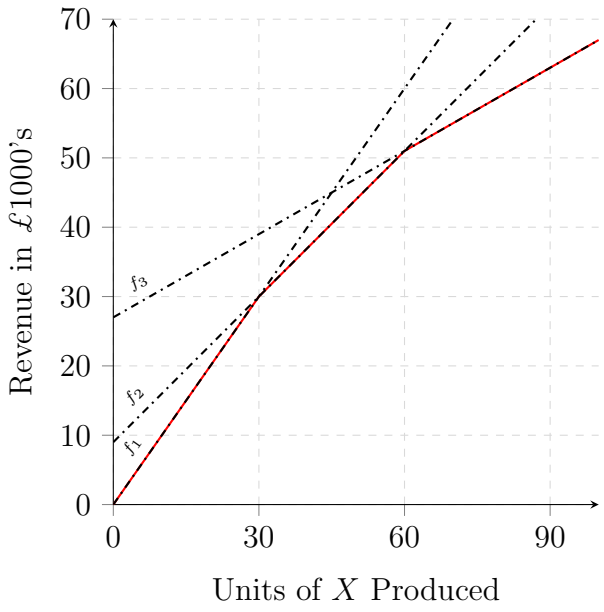


Figure 8.2: Revenue $f(x)$ (in red) for Example 9.2 and plots of $f_1(x)$, $f_2(x)$, $f_3(x)$.

and so:

$$700 \cdot 60 + 9000 = 400 \cdot 60 + b_3,$$

which we can solve to get $b_3 = 27000$. Thus:

$$f_3(x) = 400x + 27000.$$

Let's double-check our work by plotting our functions for f_1,f_2, and f_3, together with the function for f, as shown in Figure 9.2. We see that each of our lines does indeed correspond to one of the linear pieces of f. We can also see that for any x, $f(x)$ is equal to the *lowest*of our three lines. That is, $f(x) = \min(f_1(x), f_2(x), f_3(x))$. This is because f was piecewise linear and *concave*. Our goal is to maximise f, which we can now see is a minimum of 3 linear functions. To put everything together, we can then use the procedure we saw earlier for maximising the minimum of several values.

Using the procedure, since we want to maximise $f(x) = \min(f_1(x), f_2(x), f_3(x))$ we introduce a new variable (let's use z) to represent the minimum and then introduce constraints $z \leq f_1(x)$, $z \leq f_2(x)$, $z \leq f_3(x)$. Using the definitions for

each of f_1, f_2, f_3, we get:

$$
\begin{aligned}
\text{maximize} \quad & z + 1800y \\
\text{subject to} \quad & z \leq 1000x \\
& z \leq 700x + 9000 \\
& z \leq 400x + 27000 \\
& x = 4p_1 + 3p_3 + 6p_4 \\
& y = p_2 + p_3 + 3p_4 \\
& m = 100p_1 + 70p_2 + 120p_3 + 270p_4 \\
& e = 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \\
& l = 16p_1 + 16p_2 + 50p_3 + 48p_4 \\
& m \leq 6000 \\
& e \leq 100000 \\
& l \leq 1000 \\
& p_1, p_2, p_3, p_4 \geq 0 \\
& x, y, z, m, e, l \text{ unrestricted}
\end{aligned}
$$

A similar trick can be used to *minimise* a piecewise linear *convex* function f. We do exactly the same thing as above, except at the end we will find that $f(x)$ is given by the *maximum* of each of the pieces $f_1(x), f_2(x), \ldots$. Then, we can use our trick for minimising a maximum to model the problem.

Week 9

Advanced Modelling and Game Theory (I)

9.1 Advanced Modelling: Handling Min-Max Problems

Before continuing on to game theory, let's return briefly to the question of what we can model with linear programs. We will see that there are a few "tricks" that allow certain kinds of non-linear objectives to be handled with a linear program. Both tricks rely on the fact that we have an algorithm for computing the optimal solution (namely, the simplex algorithm). Specifically, we will see that even though our linear programs in the next two sections may have feasible solutions that do not correspond to the reality of the problem we are modelling, we can guarantee that this will not happen for an *optimal* solution.

For our first problem, refer back to the following example from Week 2:

Example 9.1. A factory makes 2 different parts (say, part X and part Y). Their plant has 4 separate processes in place: there are two older processes (say, process 1 and 2) that produce parts X and Y directly, as well as two different integrated processes for producing both X and Y simultaneously. The 4 processes can be run simultaneously, but require labour, raw metal, and electricity. The hourly inputs and outputs for each process are as follows:

	Outputs		Inputs		
Process	X	Y	Metal	Electricity	Labour
1	4	0	100 kg	800 kWh	16 hrs
2	0	1	70 kg	600 kWh	16 hrs
3	3	1	120 kg	2000 kWh	50 hrs
4	6	3	270 kg	4000 kWh	48 hrs

Suppose now that we wanted to know *how fast* we could produce 120 of part X and 50 of part Y, if we had *unlimited resources*. This could be used to find out, for example, the absolute minimum number of hours we need to keep the factory open to still meet our production requirements. To figure this out we need to see which of our processes runs for the *longest time*. The quantity we are interested in is thus given by the maximum of x_1, x_2, x_3, x_4. But, is this a linear function? No! For example $\max(2, 0, 0, 0) = 2$ and $\max(0, 1, 0, 0) = 1$ but:

$$\max(2+0, 0+1, 0+0, 0+0) = 2 \neq 2+1\,.$$

But, what do we know about the maximum of a set? Obviously, it must be larger than any element of the set. Recall that we introduced decision variables p_1, p_2, p_3, p_4 representing how long to run each process when modelling this problem in Week 2. Let's introduce a new variable m to represent the maximum of p_1, p_2, p_3, p_4. Then, we know that:

$$\begin{aligned} m &\geq p_1 \\ m &\geq p_2 \\ m &\geq p_3 \\ m &\geq p_4 \end{aligned}$$

These are *definitely* linear constraints. Let's add m to our production program, together with these constraints, and then just try to minimise m. Doing this we

get something like the following:

$$
\begin{aligned}
\text{minimize} \quad & m \\
\text{subject to} \quad & m \geq p_j \qquad \text{for each } j = 1, 2, 3, 4 \\
& x = 4p_1 + 3p_3 + 6p_4 \\
& y = p_2 + p_3 + 3p_4 \\
& x \geq 120 \\
& y \geq 50 \\
& m \geq p_1 \\
& m \geq p_2 \\
& m \geq p_3 \\
& m \geq p_4 \\
& p_1, p_2, p_3, p_4 \geq 0 \\
& x, y, m \text{ unrestricted}
\end{aligned}
\tag{9.1}
$$

Here, the first 2 constraints tell us how many parts X and Y we can make when we run our processes for p_1, p_2, p_3, p_4 hours, respectively. The second 2 constraints say that we must produce at least the required number of parts X and Y. The last 4 constraints say that the variable m representing the maximum of p_1, p_2, p_3, p_4 must be at least as large as each of p_1, p_2, p_3, p_4. Notice that, in general, the feasible solutions of this program will not correspond exactly to what we want, since the maximum is *equal* to the largest value of p_1, p_2, p_3, p_4, but here we only ask that m be *at least* each of these values. However, we will show that for an *optimal* solution to our program, then m will always be equal to the maximum of p_1, p_2, p_3, p_4. This is good enough for us, assuming our goal is to actually solve the problem optimally.

Intuitively, the idea is that as long as m is strict larger than the maximum of $p_1, \ldots, p_4$, we could decrease it to get a better solution for our program. Formally, consider any feasible solution to our program, and suppose that it assigns values $p_1^*, \ldots, p_4^*$ and m^* to the variables $p_1, \ldots, p_4$ and m. Then, since the solution is feasible, we must have that $p_i^* \leq m^*$ for each $i = 1, 2, 3, 4$. Suppose that $m^* > \max(p_1^*, \ldots, p_4^*)$ and set $\varepsilon = m^* - \max(p_1^*, \ldots, p_4^*) > 0$. Then, $m^* - \varepsilon = \max(p_1^*, \ldots, p_4^*) \geq p_i^*$ for each $i = 1, 2, 3, 4$, so setting $m = m^* - \varepsilon$ gives a feasible solution to the linear program—note that we only changed m and it does not occur in any other constraints, so they are still satisfied. This feasible solution also has an objective value that is ε *smaller* than our original Thus, our original solution *could not have been optimal*. It follows any optimal solution to our transformed program must have $m = \max(p_1, \ldots, p_4)$, just as we claimed.

Note that we can repeat the above procedure in any mathematical program that requires we *minimise* the *maximum* of some set of variables. Whenever we

have a mathematical *minimisation* program with an objective of the form:

$$\max(x_1, \ldots, x_k)$$

for some set of variables (here we call them $x_1, \ldots, x_k$), we do the following:

1. We change the program by introducing a new unrestricted variable (let's call it m).
2. We replace the term $\max(x_1, \ldots, x_k)$ in the objective by m.
3. We add a constraint saying that m should be larger than each of the variables in the maximum. That is, for each variable x_i that appears in the maximum, we introduce the constraint:
$$m \geq x_i$$

The above procedure will give us a linear program, by getting rid of the maximum. We can use a similar trick to model any linear program that involves *maximising* the *minimum* of several variables. There, we replace $\min(x_1, \ldots, x_k)$ by m and add constraints $m \leq x_i$ for each x_i.

9.2 Piecewise Linear Concave and Convex Objectives

Let's look at another production example, which is a modification of Example 9.1.

Example 9.2. Consider the setting from Example 9.1 and suppose that, as before, each unit of part Y sells for £1800, but that due to market demand, we can only sell the first 30 units of part X for £1000. Further units up to 60 can be sold for £700 and any excess units after that can be sold only for £400. Suppose you have 6000 kg of metal, 100000 kWh of electricity, and 1000 h of labour available. How should we schedule production to maximise revenue?

Here, the revenue from selling X is given by the function f depicted in Figure 9.1. Note that this is *not* a linear function. It is a combination of several different linear functions "glued together." We call such functions *piecewise linear*. Note that the function f behaves like a linear function f_1 passing through the origin with slope 1000 when $x \in [0, 30)$, like a linear function f_2 with slope 700 when $x \in [30, 60)$, and like a linear function f_3 with slope 400 when $x \in [60, \infty)$.

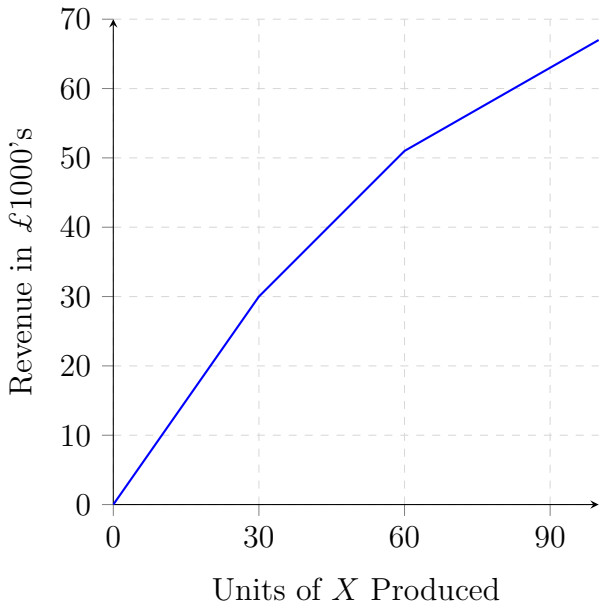


Figure 9.1: Revenue for Example 9.2

We say that the function f is *concave* because these slopes are *decreasing*. This should agree roughly with your intuition for what graphs of concave functions look like.

We cannot model this problem directly as a linear program, since our profit function is non-linear. However, we will see that since it is *piecewise* linear and *concave* we can still *maximise* it using a linear program. To do this, suppose that x is the decision variable for how many of part X we make—that is, x is the argument of our piecewise linear concave function f. We want to find an expression for each linear piece of the function f.

Let's start with the first piece f_1. We know that the slope of f_1 is 1000 and f_1 passes through the origin, so $f_1(0) = 0$. Thus, we have:

$$f_1(x) = 1000x + 0 = 1000x$$

Now, let's look at f_2. We know that the slope of f_2 is 700, so f_2 has the general form:

$$f_2(x) = 700x + b_2$$

for some constant b that we will now determine. Intuitively, b determines how much we "shift" the for f_2 up from the origin, and we want to shift it so that it joins up with the line for f_1 at the point $x = 30$, where f stops behaving like f_1 and starts behaving like f_2. Formally, since f is continuous[1], we must have

[1]If you've not encountered the formal definition of continuity: all we are saying here is that

$f_1(30) = \lim_{x\to 30^+} f(x) = \lim_{x\to 30^-} f_2(30) = f_2(30)$. Using the expression we already have for $f_1(x)$ we get:

$$f_1(30) = 1000 \cdot 30 \qquad\qquad f_2(30) = 700 \cdot 30 + b_2$$

and so:

$$1000 \cdot 30 = 700 \cdot 30 + b_2$$

which we can solve to get $b_2 = 9000$. Thus:

$$f_2(x) = 700x + 9000.$$

We can now repeat the above procedure to find an equation for f_3. Remember that $f(x)$ behaves like $f_2(x)$ for $x \in [30, 60)$ and like $f_3(x)$ for $x \in [60, \infty)$. So, now we need $f_2(60) = f_3(60)$, in order for these 2 "pieces" to join up at 60. We know the slope of f_3 is 400 and we need to find its intercept b_3. We now have:

$$f_2(60) = 700 \cdot 30 + 9000 \qquad\qquad f_3(60) = 400 \cdot 60 + b_3$$

and so:

$$700 \cdot 60 + 9000 = 400 \cdot 60 + b_3,$$

which we can solve to get $b_3 = 27000$. Thus:

$$f_3(x) = 400x + 27000.$$

Let's double-check our work by plotting our functions for f_1,f_2, and f_3, together with the function for f, as shown in Figure 9.2. We see that each of our lines does indeed correspond to one of the linear pieces of f. We can also see that for any x, $f(x)$ is equal to the *lowest*of our three lines. That is, $f(x) = \min(f_1(x), f_2(x), f_3(x))$. This is because f was piecewise linear and *concave*. Our goal is to maximise f, which we can now see is a minimum of 3 linear functions. To put everything together, we can then use the procedure we saw earlier for maximising the minimum of several values.

Using the procedure, since we want to maximise $f(x) = \min(f_1(x), f_2(x), f_3(x))$ we introduce a new variable (let's use z) to represent the minimum and then introduce constraints $z \leq f_1(x)$, $z \leq f_2(x)$, $z \leq f_3(x)$. Using the definitions for

the "piece" for f_1 must end at the same value that the "piece" for f_2 starts.

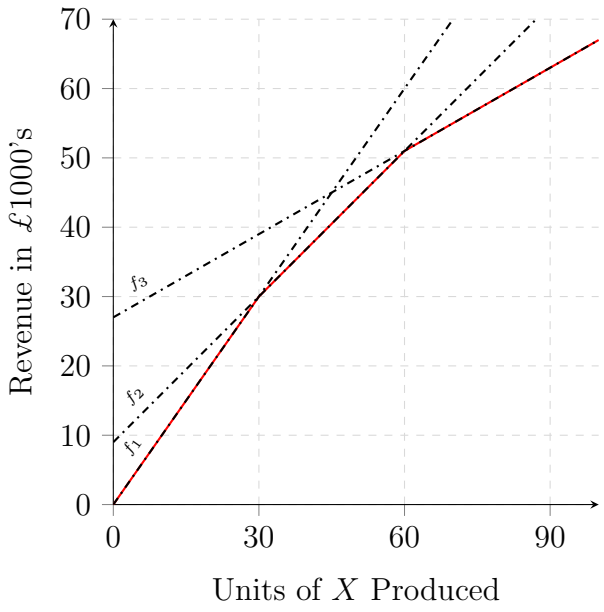


Figure 9.2: Revenue $f(x)$ (in red) for Example 9.2 and plots of $f_1(x)$, $f_2(x)$, $f_3(x)$.

each of f_1, f_2, f_3, we get:

$$
\begin{aligned}
\text{maximize} \quad & z + 1800y \\
\text{subject to} \quad & z \leq 1000x \\
& z \leq 700x + 9000 \\
& z \leq 400x + 27000 \\
& x = 4p_1 + 3p_3 + 6p_4 \\
& y = p_2 + p_3 + 3p_4 \\
& m = 100p_1 + 70p_2 + 120p_3 + 270p_4 \\
& e = 800p_1 + 600p_2 + 2000p_3 + 4000p_4 \\
& l = 16p_1 + 16p_2 + 50p_3 + 48p_4 \\
& m \leq 6000 \\
& e \leq 100000 \\
& l \leq 1000 \\
& p_1, p_2, p_3, p_4 \geq 0 \\
& x, y, z, m, e, l \text{ unrestricted}
\end{aligned}
$$

A similar trick can be used to *minimise* a piecewise linear *convex* function f. We do exactly the same thing as above, except at the end we will find that $f(x)$

is given by the *maximum* of each of the pieces $f_1(x), f_2(x), \ldots$. Then, we can use our trick for minimising a maximum to model the problem.

Now, we move on to the module's other major topic, which is game theory. Game theory concerns problems in which several independent agents or players interact. In this module, we will consider only situations in which 2 players interact. This is just a small part of the general theory, but it is enough to get some idea of several fundamental results and techniques.

9.3 Two Player Matrix Games

At its heart, game theory is about how people make decisions, and we will study situations in which several individuals must independently decide what to do. Like any mathematical theory, we must make some assumptions about our setting in order to be able to model problems and derive results. The first assumption we will make is that these individuals are *rational*—that is, each individual carefully considers her options and make the best possible decision under the circumstances. But, what do we mean by "best?"

We assume that each player has some underlying preferences about the world. These can be encoded in the form of a *utility* or *payoff* function. Formally, we will study situations in which each player i has available some set of actions A_i, and these actions lead to some set of possible outcomes O. We suppose that each individual i has a function $u_i : O \to \mathbb{R}$ representing how happy they are with the outcome O. When we talk about games, we often use the word "payoff" to talk about this value. Intuitively, you could think about $u_i(x)$ representing a monetary amount that player i receives when outcome $x \in O$ occurs. However, people often consider other factors when making decisions. For example, when offered two possible jobs, many people would rather take a job that was interesting or rewarding—or a job that didn't require commuting—even if this job payed slightly less. Luckily, we can avoid thinking about all of these possible factors by simply assuming that our players has already incorporated all this kind of information them into their functions u_i. The main property is that player is that $u_i(a)$ should be greater than $u_i(b)$ whenever player i prefers outcome a to outcome b for *whatever reason*. Sometimes a distinction is made between the notion of a payoff and a utility, but in this module, we will use the terms interchangeably.

If each player has a payoff function u_i that represents the relative value they place on certain outcomes, we can formally state our assumption that players are rational as follows:

Definition 9.1 (Theory of Rational Choice)**.** in any situation, a rational player will select an action that makes u_i as large as possible.

That is, we will assume that each player i acts to *maximise* her own utility u_i.

We will consider the following general setting. There are 2 players, each making a decision independently.

Definition 9.2 (Two-Player Strategic Game)**.** A *two player strategic game* is a game with 2 players, that we will call player 1 and player 2. It is specified by:

- A set of actions A_1 that player 1 can take. We call these *strategies* for player 1.
- A set of actions A_2 that player 2 can take. We call these *strategies* for player 2.
- A pair of *payoff functions* $u_1 : A_1 \times A_2 \to \mathbb{R}$ and $u_2 \in A_1 \times A_2 \to \mathbb{R}$. For any possible pair of strategies $a_1 \in A_1$ and $a_2 \in A_2$ that player 1 and player 2 select, $u_i(a_1, a_2)$ gives the *payoff* (or *utility*) that player i receives when player 1 chooses action a_1 and player 2 chooses action a_2.

In a 2-player strategic game, we have one outcome for each possible pair of decisions in $A_1 \times A_2$. Notice that a player's utility depends not only on her own choice but also on the *other player's* choice.

When the number of possible strategies for each player is small, we can easily represent two-player games by using a matrix. We label the rows of this matrix with one player's strategies and the columns with the other player's. In each entry in the matrix, we list the payoffs for the row player and column player. To help us keep things straight, let's name our row player Rosemary (i.e. ROW-smary) and our column player Colin (i.e. COL-in), and let R and u_r be Rosemary's set of strategies and utility function and C and u_c be Colin's set of strategies and utility function. We will also assume that the amounts won or lost by the players correspond to their payoff functions (if not, we could just rephrase the game so that each "wins" or "loses" an amount equal to however much they value the outcome). Here is an example of how we represent a game with payoff matrix:

Example 9.3. Suppose that Rosemary and Colin each have 2 cards, labelled with a 1 and a 2. Each selects a card and then both reveal their selected cards. If the sum s of the numbers on their cards is even, then Rosemary wins and Colin must pay her this s. Otherwise, Colin wins and Rosemary must pay him s.

In this case, each player has 2 strategies, corresponding to whether to choose their 1 card or 2 card. For simplicity, let's name these strategies "1" and "2". Then, the payoff matrix for this game is given by:

	1	2
1	$(2, -2)$	$(-3, 3)$
2	$(-3, 3)$	$(4, -4)$

Note that in each entry of the matrix, we always write Rosemary's payoff first and Colin's payoff second.

The game in this example has a special structure. Namely, you can see that Rosemary and Colin's payoffs sum to zero in every outcome. in other words, whatever Rosemary gains, Colin loses, and vice versa. These games model purely competitive scenarios, and are called *zero-sum* games:

Definition 9.3 (Zero-Sum Game)**.** A two-player strategic game is a *zero-sum game* if and only if for every possible outcome $(a_1, a_2) \in A_1 \times A_2$ we have $u_1(a_1, a_2) = -u_2(a_1, a_2)$. In other words, for any possible outcome, the sum of the payoffs to both players is zero.

When thinking about zero-sum games, we can save ourselves some trouble by just listing Rosemary's entries in the payoff matrix. We know that Colin's payoffs will then be the opposite of whatever appears in the matrix. In the previous example, we could have just given the payoff matrix as:

	1	2
1	2	-3
2	-3	4

Let's see a couple of other examples of zero-sum games:

Example 9.4. Suppose we consider the same game as before, except that now Rosemary has an additional card, labelled with a 3.

Now, Rosemary has an additional strategy (choosing the 3 card) and we need to add a new row to the payoff matrix. We get:

	1	2
1	2	-3
2	-3	4
3	4	-5

From Colin's perspective, the matrix now gives the amount that he *loses* in each case. Thus, we can intuitively think as follows: Rosemary wants an outcome with a matrix entry as large as possible, and Colin wants an outcome with a matrix entry number as *small* as possible. Of course, we should keep in mind that both are always trying to make their actual payoffs as large as possible, this is just a side effect of the way we have represented these payoffs in the matrix!

Here is an example of a game with more complicated sets of strategies:

Example 9.5. Rosemary and Colin each have a £1 and a £2 coin. They each select one of them and hold it in their hand, then Colin calls out "even" or "odd" and they reveal their coins. Let s be the sum of the values of the coins. If Colin correctly guessed whether s was even or odd, he wins both coins. Otherwise, Rosemary wins both coins.

Here, Rosemary has only 2 strategies, choosing a £1 or £2 coin. Colin, on the other hand, must choose a coin *and* a value in $\{\text{even}, \text{odd}\}$. We can represent her strategies as pairs of choices—a choice of coin and a choice of even/odd. Each player starts with £3, and their payoff from the game represents how much money they win or lose. The payoff matrix then looks like this:

	$(1, \text{odd})$	$(2, \text{odd})$	$(1, \text{even})$	$(2, \text{even})$
1	1	-1	-1	2
2	-2	2	1	-2

9.4 Rational Strategies

Now that we know how to model simple purely competitive situations, let's turn to the problem of analysing how a rational agent should play. Let's start with a simple example. Suppose we have a 2-player game with the following payoff matrix:

	c_1	c_2
r_1	100	-50
r_2	8	20

How should Rosemary decide on a strategy? Since she wants to maximise her utility, one choice would be to simply find the strategy that leads to the largest potential payoff. Here, if she chooses strategy r_1, she can get the largest possible payoff of 100. But, this happens *only if* Colin chooses strategy c_1. However, if Colin plays c_2 Rosemary will have the worst possible outcome! This suggests that simply picking the strategy with the largest possible payoff is a bad choice. She

could try instead to sum up all the values in each row. Notice, however, that this would still lead her to choose r_1, which certainly seems like a bad idea.

Instead, Rosemary thinks about things for a bit, and realises that no matter what strategy she adopts, she cannot control what Colin does. Thus, she should instead worry about the *worst* case that happens for each of her strategies. Formally, we call this quantity the *security level* for a strategy.

Definition 9.4 (Security Level)**.** For the row player (Rosemary), the security level of any strategy $r_i \in R$ is defined as:

$$\min_{c_j \in C} u_r(r_i, c_j)$$

Similarly, for the column player (Colin), the security level of any strategy $c \in C$ is defined as:

$$\min_{r_i \in R} u_c(r_i, c_j)$$

Our first principle is that a rational player should always act to *maximise* his her security level. For Rosemary, the corresponding payoff according to this goal can be written formally as:

$$\max_{r_i \in R} \min_{c_j \in C} u_r(r_i, c_j) \, .$$

Similarly, Colin's payoff according to his goal is:

$$\max_{c_j \in C} \min_{r_i \in R} u_c(r_i, c_j) \, .$$

Notice that both players are trying to *maximise a minimum*—that is, they want to make a choice whose worst possible outcome is as good as possible.

In a zero-sum game presented as a payoff matrix, it is easier to identify each strategy r_i with row i of the payoff matrix A and each strategy c_j with the jth column of the payoff matrix A and then think about the matrix entries. For any r_i and c_j we have $u_r(r_i, c_j) = a_{i,j} = -u_c(r_i, c_j)$. Then, we can rewrite our definition of security level using only the entries of the payoff matrix as follows. For any strategy $r_i \in R$ the corresponding security level for Rosemary will be:

$$\min_{c_j \in C} a_{i,j} \, .$$

Similarly for any $c_j \in C$, the corresponding security level for Colin will be:

$$\max_{r_i \in R} a_{i,j} \, .$$

Of course, the second quantity is actual -1 times Colin's utility (and security level), but when thinking about zero-sum games, it is useful to fix a player (here, Rosemary) and only think of her utilities. The numbers we work with are thus the payments to Rosemary, so it makes sense that Colin wants to make these as *small* as possible. From now on, we'll adopt this convention when talking about security levels and utilities (that is, all are given in terms of payments *to Rosemary*).

The principle that rational players act to maximise security tells us something about how the player's should behave. In our example game, Rosemary's security level for r_1 is -50 and her security level for r_2 is 8. Similarly, Colin's security level for strategy c_1 is 100 and his security level for c_2 is 20 (remember, though, these really correspond to *losing* 100 and 20 units from Colin's perspective). This leads to an interesting problem. If Colin suspects that Rosemary is going to play to maximise her security level, he would do better by playing strategy c_2. Anticipating this, however, Rosemary might play r_1. We can keep going around like this in circles forever.

Instead, let's just note that we expect that a pair of rational players' strategies should tend to points that are *equilibrium*. To keep things simple, let's just give the definition in terms of the payoff matrix. Later, we will see a more general definition.

Definition 9.5 ((Pure) Nash Equilibrium for a Zero-Sum Game)**.** Consider a zero-sum game with payoff matrix A. A pair of strategies $r_i \in R$ and $c_j \in C$ are said to be a *Nash equilibrium* for this game if and only if both:

$$a_{i,j} \geq a_{i',j} \text{ for all } r_{i'} \in R$$
$$a_{i,j} \leq a_{i,j'} \text{ for all } c_{j'} \in C$$

A Nash Equilibrium (r_i, c_j) represents a steady-state for a two-player game. Suppose that Rosemary plays r_i and Colin plays c_j. Then, this is Nash equilibrium if and only if, under the assumption that Colin continues playing strategy c_j, there is no incentive for Rosemary to change her strategy to anything other than r_i and similarly, under the assumption that Rosemary continues playing strategy r_i, there is no incentive for Colin to change his strategy to anything other than c_j.

Looking at our previous example games, you may be concerned that most of them don't seem to have any such equilibrium points. We'll return to this question in the next lecture. For now, however, we establish a necessary and sufficient condition for a game to possess a pair of strategies (s_i, c_j) that is a Nash equilibrium. First, let's see an example where an equilibrium does exist.

Example 9.6. Suppose we seek a pair of strategies (r_i, c_j) that form a Nash equilibrium for the game with the following payoff matrix:

	c_1	c_2	c_3	c_4	c_5
r_1	2	-3	-3	12	-5
r_2	2	7	2	9	11
r_3	-1	4	0	1	0
r_4	-3	5	1	2	-3

We find Rosemary's security levels by computing the minimum value in each row. We get $-5, 2, -1, -3$ for r_1, r_2, r_3, r_4, respectively. Colin's security levels are given by the maximum value of each column. We get $2, 7, 2, 12, 11$ for c_1, c_2, c_3, c_4, c_5 respectively. Thus, if both players seek to optimise their security levels (Rosemary to maximise and Colin to minimise), we expect that Rosemary should play strategy r_2 and Colin should play either strategy c_1 or c_3. Let's check to see if these values are a Nash Equilibrium. Indeed, we find that at (r_2, c_1) is at least as good for Rosemary as any other option (r_i, c_1) and at least as good for Colin as any other option (r_2, c_j). The same is true for (r_2, c_3). Thus both of these pairs of strategies are indeed Nash equilibria.

In this example, we saw that Rosemary's best (maximum) security level was equal to Colin's best (minimum) security level. In terms of the matrix, we saw that the entries for (r_2, c_1) and (r_2, c_3) both had the property that they were smaller than any other number in their row (and so Colin cannot do better by deviating) and larger than any other number in their column (and so Rosemary cannot do better by deviating). This condition ends up being both necessary and sufficient for a point to be a Nash equilibrium (although shortly we will see a generalisation of strategies that allows every zero-sum game to have an equilibrium).

Theorem 9.1. Let u_r^* and u_c^* be the best security levels for the row and column players, respectively, in some two-player zero-sum game. Then, this game has a pair of strategies (r_i, c_j) that together form a Nash equilibrium if and only if $u_r^* = u_c^*$.

Proof. Suppose that r_i is the strategy for which Rosemary attains security level u_r^* and that $a_{i,y}$ is the entry the payoff matrix such that $u_r^* = a_{i,y} = \min_{c_k \in C} a_{i,k}$. Similarly, let c_j be the strategy for which Colin attains security level u_c^* and let $a_{x,j}$ be the entry in the payoff matrix such that $u_c^* = a_{x,j} = \max_{r_\ell \in R} a_{\ell,j}$. Then, first, we note that we must have:

$$u_r^* = a_{i,y} = \min_{c_k \in C} a_{i,k} \leq a_{i,j} \leq \max_{r_\ell \in R} a_{\ell,j} = a_{x,j} = u_c^* \,. \tag{9.2}$$

Now, we note that if $u_r^* = u_c^*$, we must have that (9.2) holds with equality. Then, since all the inequalities in (9.2) must be equations, we must have:

$$a_{i,j} = \max_{r_\ell \in R} a_{\ell,j} \geq a_{i',j} \text{ for all } r_{i'} \in R$$
$$a_{i,j} = \min_{c_k \in C} a_{i,k} \leq a_{i,j'} \text{ for all } c_{j'} \in C$$

It follows that (r_i, c_j) is a Nash equilibrium for the game.

For the other direction, suppose that (r_i, c_j) is a Nash equilibrium. Then, since Rosemary has no incentive to change strategies:

$$a_{i,j} \geq a_{i',j} \text{ for all } r_{i'} \in R$$

and so $a_{i,j} = \max_{r_\ell \in R} a_{\ell,j}$. Similarly, since Colin has no incentive to change strategies:

$$a_{i,j} \leq a_{i,j'} \text{ for all } c_{j'} \in C$$

and so $a_{i,j} = \min_{c_k \in C} a_{i,k}$. It follow that both inequalities in (9.2) actually hold with equality, and thus $u_r^* = u_c^*$. □

In our example, we saw that all the Nash equilibria in our zero-sum game had the same value for Rosemary (and so also for Colin). This is also always true, but we will wait and prove it for a much more general case.

Week 10

Game Theory (II)

We have seen that in a two-player, zero-sum game, a rational player should always seek to optimise his or her security level. We also saw a condition under which such a game has a pair of *pure* strategies that together form a Nash equilibrium. We also saw that even some simple games do not have any such equilibrium. This leads us to the notion of *mixed* strategies

10.1 Mixed Strategies

Consider the following zero-sum game, called "Matching Pennies":

Example 10.1. Rosemary and Colin each have a 1p coin. Simultaneously, they place their coins on the table with either heads or tails showing. If the coins match, Rosemary wins £1 from Colin. Otherwise, Colin wins £1 from Rosemary.

Here each player has 2 strategies, which we will label h or t (for heads and tails). The payoff matrix for this game looks like this:

	h	t
h	1	-1
t	-1	1

Checking this matrix, we see that there is clearly no Nash equilibrium. For any pair of strategies with the coins different, Rosemary would prefer to flip her coin, and for any pair with the coins the same, Colin would prefer to flip his coin. This perhaps suggests that Nash equilibria are not so useful, since even a simple game like this does not possess one. In this section we will generalise our notion of

strategies. For these generalised strategies, we will be able to show that in fact *every* zero-sum two-player game has a Nash equilibrium.

It seems that no matter how complicated a method Rosemary uses to choose a strategy (head or tails), Colin would be able to win the game by using the same method and then picking the opposite (tails or heads). Intuitively, one way around this might be for Rosemary to choose a strategy *randomly*. Think of the following strategy for Rosemary: she could flip her coin so that it comes randomly heads or tails. Then, neither Rosemary nor Colin know what the outcome will be, and no matter what Colin chooses, Rosemary will win half the time, since the probability of her coin coming up the same as Colin's choice will be 1/2. This leads us to the notion of a *mixed* strategy.

Definition 10.1 (Mixed Strategy)**.** Let S be a set of strategies for a player, and let $\mathbf{x}^\mathsf{T} = (x_1, \ldots, x_{|S|})$ be a collection of numbers with $x_i \geq 0$ for all i and $\sum_i x_i = 1$. We say that $\mathbf{x}$ is a *mixed strategy* for our player. We now call the original strategies in S *pure strategies* and let $\Delta(S)$ denote the set of all mixed strategies from S.

Note that, agreeing with our previous insight, we can interpret the numbers $x_1, \ldots, x_{|S|}$ as a probability distribution, since they are non-negative and sum to 1. Intuitively, the mixed strategy $\mathbf{x} \in \Delta(S)$ corresponds to a *random* strategy in which a player plays each $s_i \in S$ with probability x_i. The outcome of a game then becomes a random variable, and we assume that a rational player should act to maximise his or her *expected payoff*. If Rosemary plays strategy $\mathbf{x} \in \Delta(R)$ and Colin plays strategy $\mathbf{y} \in \Delta(C)$, the expected payoff in a zero-sum game with payoff matrix A will be given by:

$$\mathbf{x}^\mathsf{T} A \mathbf{y} = \sum_{r_i \in R} \sum_{c_j \in C} x_i y_j a_{i,j} \, .$$

Here, x_i and y_j are the probabilities that Rosemary chooses r_i and Colin chooses c_j. Since these choices are made independently, Rosemary will receive the payoff $a_{i,j}$ with probability $x_i y_j$. Thus, the above expression does indeed represent her expected payoff (and so also the opposite of Colin's expected payoff).

We can suppose, as before, that players will act to optimise their security levels. Now, however, the notion of a security level looks more complicated. In particular, Rosemary must select a mixed strategy from $\Delta(R)$ that maximises the minimum payoff over all possible strategies in $\Delta(C)$. It turns out that each player only need to consider the other player's *pure* strategies when calculating his or her security level.

Lemma 10.1. For any mixed strategy $\mathbf{x} \in \Delta(R)$,

$$\min_{\mathbf{y}\in\Delta(C)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j}$$

Similarly, for any mixed strategy $\mathbf{y} \in \Delta(C)$,

$$\max_{\mathbf{x}\in\Delta(R)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j}$$

Proof. Both results intuitively follow from the fact that a weighted average of some payoffs is at least as large as the smallest payoff and at most as large as the largest payoff. Formally, for the first result, suppose that j^* is the index j for which:

$$\sum_{r_i \in R} x_i a_{i,j}$$

is the *smallest*. Then,

$$\begin{aligned}
\min_{\mathbf{y}\in\Delta(C)} \mathbf{x}^\mathsf{T} A\mathbf{y} &= \min_{\mathbf{y}\in\Delta(C)} \sum_{r_i \in R} \sum_{c_j \in C} y_j x_i a_{i,j} && \text{(Algebra)} \\
&= \min_{\mathbf{y}\in\Delta(C)} \sum_{c_j \in C} y_j \sum_{r_i \in R} x_i a_{i,j} && \text{(Changing order of summation)} \\
&\geq \min_{\mathbf{y}\in\Delta(C)} \sum_{c_j \in C} y_j \sum_{r_i \in R} x_i a_{i,j^*} && \text{(By our choice of } j^*) \\
&= \min_{\mathbf{y}\in\Delta(C)} \sum_{r_i \in R} x_i a_{i,j^*} && (y_j \text{ sum to 1}) \\
&= \sum_{r_i \in R} x_i a_{i,j^*} && \text{(The term inside the min does not depend on } \mathbf{y}) \\
&= \min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j}\,. && \text{(By our choice of } j^*)
\end{aligned}$$

Furthermore, we note that the pure strategy c_{j^*} can be represented as a mixed strategy $\mathbf{y}'$ with $y'_{j^*} = 1$ and $y'_j = 0$ for all $j \neq j^*$. Thus,

$$\min_{\mathbf{y}\in\Delta(C)} \mathbf{x}^\mathsf{T} A\mathbf{y} \leq \mathbf{x}^\mathsf{T} A\mathbf{y}' = \min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j}\,.$$

It follows that $\min_{\mathbf{y}\in\Delta(C)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j}$.

Similarly, for the second claim, let i^* be the index for which

$$\sum_{c_j \in C} y_j a_{i,j}$$

is the *largest*. Then,

$$\begin{aligned}
\max_{\mathbf{x}\in\Delta(R)} \mathbf{x}^\mathsf{T} A\mathbf{y} &= \max_{\mathbf{x}\in\Delta(R)} \sum_{r_i \in R} \sum_{c_j \in C} y_j x_i a_{i,j} && \text{(Algebra)} \\
&\leq \max_{\mathbf{x}\in\Delta(R)} \sum_{r_i \in R} x_i \sum_{r_i \in R} y_j a_{i^*,j} && \text{(By our choice of } i^*) \\
&= \max_{\mathbf{x}\in\Delta(R)} \sum_{c_j \in C} y_j a_{i^*,j} && (x_i \text{ sum to 1}) \\
&= \sum_{c_j \in C} y_j a_{i^*,j} && \text{(The term inside the max does not depend on } \mathbf{x}) \\
&= \max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j} && \text{(By our choice of } i^*)
\end{aligned}$$

Further, we note that if we set $\mathbf{x}'$ so that $x'_{i^*} = 1$ and $x'_i = 0$ for all $i \neq i^*$ then:

$$\max_{\mathbf{x}\in\Delta(R)} \mathbf{x}^\mathsf{T} A\mathbf{y} \geq \mathbf{x}'^\mathsf{T} A\mathbf{y} = \max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j},$$

so indeed $\max_{\mathbf{x}\in\Delta(R)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j}$. □

We want Rosemary and Colin's security levels for a given mixed strategy to represent the worst possible outcome that can happen for each of them when they play this strategy. The previous theorem implies that this worst case will always be attained when the other player's simply plays one of their pure strategies. Indeed, note that our theorem shows that if Rosemary picks a strategy $\mathbf{x} \in \Delta(R)$, then for any of Colin's strategies in $\mathbf{y} \in \Delta(C)$ there is a there is a single pure strategy $c_j \in C$ that would be as bad or worse for Rosemary as $\mathbf{x}$. Similarly, if Colin mixed strategy $\mathbf{y} \in \Delta(C)$, then for any mixed strategy $\mathbf{x} \in \Delta(R)$ there is a single pure strategy $r_i \in R$ for Rosemary that would be as bad or worse for Colin. Based on these observations, we can state a simple formula for the security levels for both Rosemary and Colin:

Definition 10.2 (Security Level for a Mixed Strategy)**.** For any mixed strategy $\mathbf{x} \in \Delta(R)$, Rosemary's security level for $\mathbf{x}$ is given by:

$$\min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j}$$

This is the smallest expected payoff (to Rosemary) of any single column under the probability distribution $\mathbf{x}$ on rows of A.

Similarly, for any mixed strategy $\mathbf{y} \in \Delta(C)$, Colin's security level for $\mathbf{y}$ is given by:

$$\max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j}$$

This is the largest expected payoff (to Rosemary) of any single row under the probability distribution $\mathbf{y}$ on columns of A.

Next, we show that many of the results and definitions for pure strategies can be generalised in a straightforward way to handle mixed strategies. Moreover, we will see that *every* zero-sum game has a Nash equilibrium when players can play mixed strategies.

10.2 General Nash Equilibria

Our notion of equilibria for mixed strategies looks exactly like that for pure strategies—at an equilibrium, neither player has any incentive to deviate from their current strategy. However, in the general definition, we will now players allow players to use *mixed* strategies and they will consider their *expected* payoffs with respect to these mixed strategies.

Definition 10.3 (General Nash Equilibrium for a Zero-Game)**.** Consider a zero-sum game with payoff matrix A. A pair of mixed strategies $\mathbf{x} \in \Delta(R)$ and $\mathbf{y} \in \Delta(C)$ are said to be a *Nash equilibrium* for this game if and only if both:

$$\mathbf{x}^\mathsf{T} A \mathbf{y} \geq \mathbf{x}'^\mathsf{T} A \mathbf{y} \text{ for all } \mathbf{x}' \in \Delta(R)$$
$$\mathbf{x}^\mathsf{T} A \mathbf{y} \leq \mathbf{x}^\mathsf{T} A \mathbf{y}' \text{ for all } \mathbf{y}' \in \Delta(C)$$

Note that the first condition says that—assuming Colin continues to play $\mathbf{y}$—Rosemary cannot improve her expected payoff by selecting any other mixed strategy $\mathbf{x}'$. Similarly, the second condition says that—assuming Rosemary continues

to play $\mathbf{x}$—Colin cannot improve his expected payoff by selecting any other mixed strategy $\mathbf{y}'$.

We now give a generalisation of Theorem 9.1 that will allow us to characterise when a pair of mixed strategies forms a Nash equilibrium:

Theorem 10.2. Let $\mathbf{x} \in \Delta(R)$ and $\mathbf{y} \in \Delta(C)$ be a pair of mixed strategies for a two-player, zero-sum game, and let:

$$u_r^* = \min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j}$$
$$u_c^* = \max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j}$$

be the security levels for $\mathbf{x}$ and $\mathbf{y}$, respectively. Then, $(\mathbf{x}, \mathbf{y})$ is a Nash equilibrium for this game if and only if $u_r^* = u_c^*$.

Proof. Similarly to the proof of Theorem 9.1, we must have:

$$\min_{c_j \in C} \sum_{r_i \in R} x_i a_{i,j} = \min_{\mathbf{z} \in \Delta(C)} \mathbf{x}^\mathsf{T} A \mathbf{z} \leq \mathbf{x}^\mathsf{T} A \mathbf{y} \leq \max_{\mathbf{z} \in \Delta(R)} \mathbf{z}^\mathsf{T} A \mathbf{y} = \max_{r_i \in R} \sum_{c_j \in C} y_j a_{i,j} \tag{10.1}$$

Here, the first and last equations follow from Lemma 10.1. Now, suppose that $u_c^* = u_r^*$. Then the right and left of (10.1) are equal and so both inequalities in (10.1) must in fact be equations. In particular, we must have that:

$$\mathbf{x}^\mathsf{T} A \mathbf{y} = \min_{\mathbf{z} \in \Delta(C)} \mathbf{x}^\mathsf{T} A \mathbf{z} \leq \mathbf{x}^\mathsf{T} A \mathbf{y}'$$

for any $\mathbf{y}' \in \Delta(C)$ and also

$$\mathbf{x}^\mathsf{T} A \mathbf{y} = \max_{\mathbf{z} \in \Delta(R)} \mathbf{z}^\mathsf{T} A \mathbf{y} \geq \mathbf{x}'^\mathsf{T} A \mathbf{y}$$

for any $\mathbf{x}' \in \Delta(R)$. Thus, $(\mathbf{x}, \mathbf{y})$ is a Nash equilibrium.

Conversely, suppose that $(\mathbf{x}, \mathbf{y})$ is a Nash equilibrium. Then, for every $\mathbf{x}' \in \Delta(R)$ we have:

$$\mathbf{x}^\mathsf{T} A \mathbf{y} \geq \mathbf{x}'^\mathsf{T} A \mathbf{y}$$

and so $\mathbf{x}^\mathsf{T} A \mathbf{y} = \max_{\mathbf{x}' \in \Delta(R)} \mathbf{x}'^\mathsf{T} A \mathbf{y}$. Similarly, for every $\mathbf{y}' \in \Delta(C)$ we have:

$$\mathbf{x}^\mathsf{T} A \mathbf{y} \leq \mathbf{x}^\mathsf{T} A \mathbf{y}'$$

and so $\mathbf{x}^\mathsf{T} A \mathbf{y} = \min_{\mathbf{y}' \in \Delta(C)} \mathbf{x}'^\mathsf{T} A \mathbf{y}$. Thus, both inequalities in (10.1) hold with equality. It follows the left and right of (10.1) are equal, and so $u_r^* = u_c^*$. □

10.3 Security levels and linear programming

So far, we have focused on characterising Nash equilibria, but have not considered how a player might go about *finding* a mixed strategy that maximises his or her security level. We will now show that in fact both players can compute their optimal security levels, as well as the corresponding strategies, by using a linear program!

Suppose that Rosemary has n pure strategies and Colin has m pure strategies, so $|R| = n$, $|C| = m$.

Let's consider Colin first. In order to optimise his security level, Colin wants to find a strategy that *minimises* the *maximum* payoff that he could have to make to Rosemary. That is, he wants to find a probability distribution $y_1 \ldots, y_m$ over his m strategies so that

$$\max_{r_i \in R} (y_1 a_{i,1} + y_2 a_{i,2} + \ldots + y_m a_{i,m})$$

is minimised. Also, for $\mathbf{y}$ to be a proper distribution, he should have:

$$y_1 + y_2 + \cdots + y_m = 1$$

and also $y_j \geq 0$ for all $1 \leq j \leq m$. is as small as possible. Formally, we can write this as a mathematical program:

$$\begin{aligned} \text{minimize} \quad & \max_{r_i \in R} (y_1 a_{i,1} + y_2 a_{i,2} + \ldots + y_m a_{i,m}) \\ \text{subject to} \quad & y_1 + \cdots + y_m = 1 \\ & y_1, \ldots, y_m \geq 0 \end{aligned}$$

Using a trick we saw in Week 3, we can rewrite this into a linear program. We introduce an unrestricted variable s to represent the minimum. Then, we say that we want to minimise s, subject to the constraints that s is at least as large as each possible expected payoff $y_1 a_{i,1} + y_2 a_{i,2} + \cdots + y_m a_{i,m}$. Formally, we get a program:

$$\begin{aligned} \text{minimize} \quad & s \\ \text{subject to} \quad & s \geq y_1 a_{1,1} + y_2 a_{1,2} + \cdots + y_m a_{1,m} \\ & s \geq y_1 a_{2,1} + y_2 a_{2,2} + \cdots + y_m a_{2,m} \\ & \vdots \\ & s \geq y_1 a_{n,1} + y_2 a_{n,2} + \cdots + y_m a_{n,m} \\ & y_1 + y_2 + \cdots + y_m = 1 \\ & y_1, \ldots, y_m \geq 0 \\ & s \text{ unrestricted} \end{aligned}$$

We can rearrange the constraints to get:

$$\begin{aligned}
\text{minimize} \quad & s \\
\text{subject to} \quad & s - y_1 a_{1,1} - y_2 a_{1,2} - \cdots - y_m a_{1,m} \geq 0 \\
& s - y_1 a_{2,1} - y_2 a_{2,2} - \cdots - y_m a_{2,m} \geq 0 \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
& s - y_1 a_{n,1} - y_2 a_{n,2} - \cdots - y_m a_{n,m} \geq 0 \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_1 + y_2 + \cdots + y_m = 1 \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_1, \ldots, y_m \geq 0 \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad s \text{ unrestricted}
\end{aligned} \tag{10.2}$$

If Colin solves this linear program, the value assigned to s will tell him his best possible security level, and the values $y_1, \ldots, y_m$ will tell him the probabilities with which he should play pure strategies $c_1, \ldots, c_m$ to attain this security level.

Let's now consider Rosemary's security level. She wants to find a distribution $\mathbf{x}$ that *maximises* the *minimum* expected payoff she might receive. That is, she wants to find a probability distribution $x_1 \ldots, x_n$ over her n strategies so that

$$\min_{c_j \in C} (x_1 a_{1,j} + x_2 a_{2,j} + \ldots + x_n a_{n,j})$$

is maximised. Similarly to Colin, for $\mathbf{x}$ to be a proper distribution, she should have:

$$x_1 + x_2 + \cdots + x_n = 1$$

and also $x_i \geq 0$ for all $1 \leq i \leq m$. We can write this problem as a mathematical optimisation program:

$$\begin{aligned}
\text{maximize} \quad & \min_{c_j \in C} (x_1 a_{1,j} + x_2 a_{2,j} + \ldots + x_n a_{n,j}) \\
\text{subject to} \quad & x_1 + \cdots + x_n = 1 \\
& x_1, \ldots, x_n \geq 0
\end{aligned}$$

We can use essentially the same trick, and introduce a new variable t to represent the minimum in the objective. Then, we need to make sure that:

$$t \leq x_1 a_{1,j} + x_2 a_{2,j} + \ldots + x_n a_{n,j}$$

for every j. We get the linear program:

$$\begin{aligned}
&\text{maximize} \quad t \\
&\text{subject to} \qquad t \le x_1a_{1,1} + x_2a_{2,1} + \ldots + x_na_{n,1} \\
&\qquad\qquad\qquad\quad t \le x_1a_{1,2} + x_2a_{2,2} + \ldots + x_na_{n,2} \\
&\qquad\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad\quad t \le x_1a_{1,m} + x_2a_{2,m} + \ldots + x_na_{n,m} \\
&\qquad\qquad x_1 + x_2 \cdots + x_n = 1 \\
&\qquad\qquad\quad x_1, \ldots, x_n \ge 0 \\
&\qquad\qquad\qquad\quad t \text{ unrestricted}
\end{aligned}$$

We can rearrange this to get a program that looks like:

$$\begin{aligned}
&\text{maximize} \quad t \\
&\text{subject to} \qquad t - x_1a_{1,1} - x_2a_{2,1} - \ldots - x_na_{n,1} \le 0 \\
&\qquad\qquad\qquad\quad t - x_1a_{1,2} - x_2a_{2,2} - \ldots - x_na_{n,2} \le 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad\quad t - x_1a_{1,m} - x_2a_{2,m} - \ldots - x_na_{n,m} \le 0 \\
&\qquad\qquad\qquad\qquad\qquad x_1 + x_2 \cdots + x_n = 1 \\
&\qquad\qquad\qquad\qquad\quad x_1, \ldots, x_n \ge 0 \\
&\qquad\qquad\qquad\qquad\qquad\quad t \text{ unrestricted}
\end{aligned} \tag{10.3}$$

If Rosemary solves this linear program, the value assigned to t will give her best possible security level, and the values $x_1, \ldots, x_n$ will tell her the probabilities with which she should play pure strategies $r_1, \ldots, r_n$ to attain this security level.

Example 10.2. Give a linear program for finding the row player's optimal mixed strategy for the zero-sum game with the following payoff matrix:

	1	2
1	2	−3
2	−3	4
3	4	−5

Solution. Since we are interested in Rosemary's optimal strategy, we should have a program with a variable x_i for each pure strategy $r_i \in R$. Our program will have a constraint saying that the expected payoff when Rosemary plays $\mathbf{x}$ should be at

most the value this strategy gives for each of Colin's pure strategies $c_j \in C$. That is, we will get a constraint for each *column* of the payoff matrix. The program is given by:

$$\begin{aligned} \text{maximize} \quad & t \\ \text{subject to} \quad & t \leq 2x_1 - 3x_2 + 4x_3 \\ & t \leq -3x_1 + 4x_2 - 5x_3 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0 \\ & t \text{ unrestricted} \end{aligned}$$

□

10.4 The Minimax Theorem

We are now prove the von Neumann minimax theorem, which shows that *every* 2-player zero-sum game has a genaral Nash equilibrium. The theorem actually applies to a fairly broad class of linear optimisation problems, but we will state it here in the language of zero-sum games.

In lectures we did not state the theorem in this way but we used essentially the same proof to show that every 2-player zero-sum game has a general (mixed) Nash equilibrium.

Theorem 10.3 (von Neumann Minimax Theorem)**.** In any zero-sum game with payoff matrix given by A, and row and column player strategies given by R and C, respectively,

$$\max_{\mathbf{x}\in\Delta(R)} \min_{\mathbf{y}\in\Delta(C)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \min_{\mathbf{y}\in\Delta(C)} \max_{\mathbf{x}\in\Delta(R)} \mathbf{x}^\mathsf{T} A\mathbf{y}\,.$$

Proof. Intuitively, the left-hand side represents the optimal security level for Rosemary, and the right-hand side represents the optimal security level for Colin. As we showed in Lemma 10.1, we can compute these by considering only pure strategies for the other player. That is, we have:

$$\max_{\mathbf{x}\in\Delta(R)} \min_{\mathbf{y}\in\Delta(C)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \max_{\mathbf{x}\in\Delta(R)} \min_{c_j\in C} \sum_{r_i\in R} x_i a_{i,j} \tag{10.4}$$

$$\min_{\mathbf{y}\in\Delta(C)} \max_{\mathbf{x}\in\Delta(R)} \mathbf{x}^\mathsf{T} A\mathbf{y} = \min_{\mathbf{y}\in\Delta(C)} \max_{r_i\in R} \sum_{c_j\in C} y_j a_{i,j} \tag{10.5}$$

Now, we will use the fact that each player can compute these security levels with a linear program. Specifically, we saw that the objective of the linear program (10.2) is equal to (10.4) and the objective of the linear program (10.3) is equal to (10.5).

Now, let's look at the dual of the linear program (10.3). Recall that this program is given by:

$$\begin{aligned}
&\text{maximize} && t \\
&\text{subject to} && t - x_1 a_{1,1} - x_2 a_{2,1} - \ldots - x_n a_{n,1} \leq 0 \\
& && t - x_1 a_{1,2} - x_2 a_{2,2} - \ldots - x_n a_{n,2} \leq 0 \\
& && \vdots \\
& && t - x_1 a_{1,m} - x_2 a_{2,m} - \ldots - x_n a_{n,m} \leq 0 \\
& && x_1 + x_2 \cdots + x_n = 1 \\
& && x_1, \ldots, x_n \geq 0 \\
& && t \text{ unrestricted}
\end{aligned}$$

Let's name the dual variables for each of the inequalities $y_1, \ldots, y_m$ and the dual variable for the equation constraint s. Then, in our dual program, we will have $y_1, \ldots, y_m \geq 0$ and s unrestricted. The objective function for our dual program will then be simply s (since right-hand side of every constraint is 0 except for the equation, which has right-hand side 1). For each non-negative primal variable x_i we will get a corresponding dual inequality constraint of the form:

$$-a_{i,1} y_1 - a_{i,2} y_2 - \ldots - a_{i,m} y_m + s \geq 0$$

and for the unrestricted primal variable t we will get a corresponding dual equation constraint:

$$y_1 + y_2 + \cdots + y_m = 1$$

Altogether, we can write the dual program as:

$$\begin{aligned}
&\text{minimize} && s \\
&\text{subject to} && s - y_1 a_{1,1} - y_2 a_{1,2} - \cdots - y_m a_{1,m} \geq 0 \\
& && s - y_1 a_{2,1} - y_2 a_{2,2} - \cdots - y_m a_{2,m} \geq 0 \\
& && \vdots \\
& && s - \geq y_1 a_{n,1} - y_2 a_{n,2} - \cdots - y_m a_{n,m} \geq 0 \\
& && y_1 + y_2 + \cdots + y_m = 1 \\
& && y_1, \ldots, y_m \geq 0 \\
& && s \text{ unrestricted}
\end{aligned}$$

Notice that this is precisely the program (10.2) for Colin's security level. By strong duality, these programs must have the same optimal solution, and so indeed the expressions (10.4) and (10.5) are equal. □

An immediate consequence of the minimax theorem shows that for any zero-sum game there is always a pair of mixed strategies for Rosemary and Colin that give the same security level. Combining this with Theorem 10.2, we get the following corollary:

Corollary 10.4. Every two-player zero-sum game has a Nash equilibrium.

The optimal security level for both players in a zero-sum game can be computed by linear programming. We have just seen that there is always a mixed strategy that gives both the same security level, which is also the expected payoff for their strategies. It could be that there is more than 1 pair of such strategies, however. Intuitively, we should expect that they all have the same expected payoff, since they are all solutions of the same pair of dual linear programs. In the following theorem, we show this directly. Additionally, we show that we can combining Rosemary and Colin's strategies from 2 different Nash equilibria to get another Nash equilibrium.

Theorem 10.5. Suppose that $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}', \mathbf{y}')$ is a pair of strategies that are both Nash equilibria for a given zero-sum game. Then, $(\mathbf{x}, \mathbf{y}')$ and $(\mathbf{x}', \mathbf{y})$ are also Nash equilibria. Moreover, all these equilibria have the same expected payoff.

Proof. Since $(\mathbf{x}, \mathbf{y})$ is a Nash equilibrium, Rosemary and Colin do not have any incentive to change to any other strategy (in particular, $\mathbf{x}'$ or $\mathbf{y}'$), and we must have both:

$$\mathbf{x}^\mathsf{T} A\mathbf{y} \geq \mathbf{x}'^\mathsf{T} A\mathbf{y}$$
$$\mathbf{x}^\mathsf{T} A\mathbf{y} \leq \mathbf{x}^\mathsf{T} A\mathbf{y}'$$

Similarly, since $(\mathbf{x}', \mathbf{y}')$ is a Nash equilibrium Rosemary and Colin do not have any incentive to change to $\mathbf{x}$ or $\mathbf{y}$. Thus, we have both:

$$\mathbf{x}'^\mathsf{T} A\mathbf{y}' \geq \mathbf{x}^\mathsf{T} A\mathbf{y}'$$
$$\mathbf{x}'^\mathsf{T} A\mathbf{y}' \leq \mathbf{x}'^\mathsf{T} A\mathbf{y}$$

Combining these 4 inequalities we get:

$$\mathbf{x}^\mathsf{T} A\mathbf{y} \;\geq\; \mathbf{x}'^\mathsf{T} A\mathbf{y} \;\geq\; \mathbf{x}'^\mathsf{T} A\mathbf{y}' \;\geq\; \mathbf{x}^\mathsf{T} A\mathbf{y}' \;\geq\; \mathbf{x}^\mathsf{T} A\mathbf{y}$$

Clearly, this can only be true if all four inequalities are tight. Thus, each of the 4 pairs of strategies $(\mathbf{x}, \mathbf{y})$, $(\mathbf{x}', \mathbf{y})$, $(\mathbf{x}, \mathbf{y}')$, $(\mathbf{x}', \mathbf{y}')$ have the same expected payoff. Let's now show that both $(\mathbf{x}', \mathbf{y})$ and $(\mathbf{x}, \mathbf{y}')$ are also Nash equilibria. Indeed, for any other strategies $\mathbf{a} \in \Delta(R)$ and $\mathbf{b} \in \Delta(C)$ we must have:

$$\mathbf{x}^\mathsf{T} A \mathbf{y}' = \mathbf{x}'^\mathsf{T} A \mathbf{y}' \geq \mathbf{a}^\mathsf{T} A \mathbf{y}'$$
$$\mathbf{x}^\mathsf{T} A \mathbf{y}' = \mathbf{x}^\mathsf{T} A \mathbf{y} \leq \mathbf{x}^\mathsf{T} A \mathbf{b}$$

Since $(\mathbf{x}', \mathbf{y}')$ and $(\mathbf{x}, \mathbf{y})$ are Nash equilibria. Thus, $(\mathbf{x}, \mathbf{y}')$ is also a Nash equilibrium. Similarly,

$$\mathbf{x}'^\mathsf{T} A \mathbf{y} = \mathbf{x}^\mathsf{T} A \mathbf{y} \geq \mathbf{a}^\mathsf{T} A \mathbf{y}$$
$$\mathbf{x}'^\mathsf{T} A \mathbf{y} = \mathbf{x}'^\mathsf{T} A \mathbf{y}' \leq \mathbf{x}'^\mathsf{T} A \mathbf{b}$$

and so $(\mathbf{x}', \mathbf{y})$ is a Nash equilibrium. □

Definition 10.4 (Value of a Two-Player Zero-Sum Game)**.** We call the common expected payoff of all Nash equilibria for a zero-sum game the *value* of the game.

Week 11

Game Theory (III)

11.1 General 2-Player Games

We now move beyond the simple setting of zero-sum games and consider arbitrary 2-player games. Recall that in a 2-player game, each possible pair of strategies $(r_i, c_j) \in R \times C$ results in a payoff $u_r(r_i, c_j)$ to Rosemary and a payoff $u_c(r_i, c_j)$ to Colin. For the past few lectures, we have always assumed that $u_c(r_i, c_j) = -u_r(r_i, c_j)$ for all $r_i \in R$ and $c_j \in C$. This allowed us to represent the game with a *single* payoff matrix, A where which we recorded $u_r(r_i, c_j)$ in the entry $a_{i,j}$. Then, we imagined Rosemary choosing her actions to make the selected outcome's entry in A as large as possible and Colin choosing his actions to make this entry as small as possible. In the case of Colin, this was still the same as *maximising* payoff, because $u_c(r_i, c_j) = -a_{i,j}$ and so becomes larger when $a_{i,j}$ becomes smaller.

Now, we return to the general setting in which both players' payoffs can be arbitrary. In order to represent such games, we need to list both players' payoffs in each possible outcome. As we did when we originally introduced 2-player games, we can do this by constructing a payoff matrix with a *pair* of payoffs in each entry. In row i, column j, we list the pair

$$(u_r(r_i, c_j), u_c(r_i, c_j))$$

The first entry is the payoff to Rosemary when she plays r_i and Colin pays c_j, and the second entry is the payoff to Colin in this same outcome. Note that we adopt the convention that we always list Rosemary's payoff first and Colin's second.

Example 11.1. Suppose that Rosemary and Colin are working on a joint project. Each of them can choose to "work hard" or "goof off." Both of them must work hard together to receive a high mark for the project. Both have utility 3 for receiving a high mark utility 1 for goofing off (regardless of what mark they receive) and utility 0 for working hard but not receiving a high mark. Give the payoff matrix for this game.

Solution. Here, there is only one outcome (work hard, work hard) that gets a high mark. In this case, both players receive payoff 3. Otherwise, the project will not receive a high mark, and a player will receive a utility value of 1 if he/she goofed off, and 0 if he/she worked hard. We can represent this as a matrix as follows:

	work hard	goof off
work hard	$(3, 3)$	$(0, 1)$
goof off	$(1, 0)$	$(1, 1)$

□

Notice that our example game is *not* a zero-sum game, because the pair of payoffs do not sum to zero in every case. This means that the game is no longer strictly competitive: An outcome that makes Rosemary happy does not necessarily make Colin less happy. In this scenario, we can see that each player receives the highest payoff when they work hard *together*. Do we expect that this will always happen?

Unfortunately, a lot of our intuition about rational players stops working for non-zero-sum games. In particular, we can no longer compute optimal strategies by using linear programming! Luckily, we can still use the notion of an equilibrium state to reason about these games.

Definition 11.1 ((Pure) Nash Equilibrium for a general two-player game)**.** Consider a general two player game in which the row and column player have strategies R and C and utility functions $u_r : R \times C \to \mathbb{R}$ and $u_c : R \times C \to \mathbb{R}$. Then, a pair of strategies $r_i \in R$ and $c_j \in C$ is a *Nash equilibrium* for this game if and only if both

$$\begin{aligned} u_r(r_i, c_j) &\geq u_r(r_{i'}, c_j) && \text{for all } r_{i'} \in R \\ u_c(r_i, c_j) &\geq u_c(r_i, c_{j'}) && \text{for all } c_{j'} \in C \end{aligned}$$

Notice that this corresponds exactly to our previous notion of an Nash equilibrium for zero-sum games. It simply says that if (r_i, c_j) is a Nash equilibrium then neither player would prefer to switch to any of his or her other strategies—assuming that

the other player keeps playing his or her current strategy. You can check that if $u_r(r_i, c_j) = a_{i,j}$ and $u_c(r_i, c_j) = -a_{i,j}$ for all i and j, then this definition indeed simplifies to Definition 9.5.

If we look at the payoffs in Example 12.1, we can identify 2 pure Nash equilibria. If both players are working hard, then neither player has an incentive to switch to the goof off strategy. If both players are goofing off, then neither player has an incentive to switch to the work hard strategy. Thus, (work hard, work hard) and (goof off, goof off) are both Nash equilibria of this game.

This demonstrates 2 major differences between zero-sum games and general games. First, notice that our 2 Nash equilibria have different payoffs. In contrast, we proved that for zero-sum games all Nash equilibria would have the same payoff to Rosemary and so also to Colin. Next, suppose that Rosemary and Colin each compute a security level, corresponding to the worst thing that could happen if they played a given strategy. Rosemary finds that her worst payoff when she works hard is 0, and her worst payoff when she goofs off is 1. Colin finds the same. Thus, both players have a security level of at most 1. However, in contrast to zero-sum games there is an equilibrium strategy that is *better* than this security level. In fact, the best Nash equilibrium gives each player their *worst* security level!

This means that thinking about the best possible strategy in a non-zero-sum game is not so straightforward. In general, our analysis may depend on whether or not we assume that players are allowed to communicate before the game to decide on a mutually beneficial strategy. In zero-sum games this was not an issue, because the players were always in competition with one another.

Instead of using security levels, we can compute pure Nash equilibria by considering Rosemary and Colin's *best response* to each others possible strategies:

Definition 11.2 (Best Response to a Strategy)**.** For any strategy $c_j \in C$, Rosemary's *best response to* c_j is any strategy $r_i \in R$ with:

$$u_r(r_i, c_j) = \max_{r_\ell \in R} u_r(r_\ell, c_j)\,.$$

For any strategy $r_i \in R$, Colin's *best response to* r_i is any strategy $c_j \in C$ with:

$$u_c(r_i, c_j) = \max_{c_k \in C} u_c(r_i, c_k)\,.$$

Intuitively, each player's best response to some opponent's strategy x is simply the strategy from their own set that gives them the highest payoff when their opponent plays strategy x. We can easily compute the best responses by looking at our matrix. Each of Colin's strategies is a column. Thus, Rosemary's best

response to a strategy c_j corresponds to the row that has the largest first entry for the column corresponding to c_j. Note that if there is a tie for the largest entry, then *all* of the tied strategies will be best responses. Similarly, Colin's best response to a strategy $r_i \in R$ corresponds to the column that has the largest second entry in the row corresponding to r_i.

Note that in order for a pair of pure strategies (r_i, c_j) to be a Nash equilibrium, Rosemary must have no better payoff in the column corresponding to Colin's current strategy c_j and Colin must have no better payoff in the row corresponding to Rosemary's current move r_i. That is, r_i must be a best response to c_j and vice versa. This gives us an easy way to find all pure Nash equilibria.

We go through the columns of the matrix one by one and mark the largest payoff for Rosemary in each column. These will be her best responses for each of Colin's corresponding strategies. Then, we go through the rows and mark the largest payoff to Colin each row. These will be his best responses for each of Rosemary's corresponding strategies. Then, a pair of strategies is a pure Nash equilibrium if and only if its payoffs to both Colin and Rosemary have been marked.

For Example 12.1, underlining Rosemary's best responses gives us:

	work hard	goof off
work hard	$(\underline{\mathbf{3}}, 3)$	$(0, 1)$
goof off	$(1, 0)$	$(\underline{\mathbf{1}}, 1)$

Then, additionally underlining Colin's best responses, we get:

	work hard	goof off
work hard	$(\underline{\mathbf{3}}, \underline{\mathbf{3}})$	$(0, 1)$
goof off	$(1, 0)$	$(\underline{\mathbf{1}}, \underline{\mathbf{1}})$

Thus, we find that the pure Nash equilibria are (word hard, work hard) and (goof off, goof off).

Example 11.2. Find all pure Nash equilibria for the games with the following payoff matrices.

	c_1	c_2	c_3
r_1	$(0, -1)$	$(1, 2)$	$(2, -1)$
r_2	$(2, 1)$	$(0, -1)$	$(2, 1)$
r_3	$(0, 2)$	$(-1, 1)$	$(1, -1)$

	c_1	c_2	c_3
r_1	$(1, 0)$	$(0, 1)$	$(1, -1)$
r_2	$(-1, 1)$	$(1, 0)$	$(0, 1)$

Solution. As before, to find the best responses, we mark the largest first value in each column and the largest second value in each row. For the first game, this

gives:

	c_1	c_2	c_3
r_1	$(0,-1)$	$(\underline{\mathbf{1}},\underline{\mathbf{2}})$	$(\underline{\mathbf{2}},-1)$
r_2	$(\underline{\mathbf{2}},\underline{\mathbf{1}})$	$(0,-1)$	$(\underline{\mathbf{2}},\underline{\mathbf{1}})$
r_3	$(0,\underline{\mathbf{2}})$	$(-1,1)$	$(1,-1)$

Thus, the pure Nash equilibria are (r_1, c_2), (r_2, c_1), and (r_2, c_3). Notice that we marked both of Rosemary's 2's in the third column and both of Colin's 1's in the second row, since both had the largest value.

For the second game, we get:

	c_1	c_2	c_3
r_1	$(\underline{\mathbf{1}},0)$	$(0,\underline{\mathbf{1}})$	$(\underline{\mathbf{1}},-1)$
r_2	$(-1,\underline{\mathbf{1}})$	$(\underline{\mathbf{1}},0)$	$(0,\underline{\mathbf{1}})$

Thus, this game does not have any pure Nash equilibria. □

Our second example showed that, just like we saw for zero-sum games, a non-zero sum game might not have any pure Nash equilibrium. The solution to this problem is the same as it was there. We can allow Rosemary and Colin to play mixed strategies $\mathbf{x} \in \Delta(R)$ and $\mathbf{y} \in \Delta(C)$. Then we examine each of their expected payoffs under these strategies. Let A_r be the matrix that contains only Rosemary's payoffs and A_c be the matrix that contains only Colin's payoffs (that is, A_r has as its entries the first number from each cell in our combined payoff matrix and A_c has as its entries the second number) then, Rosemary's expected payoff will be $\mathbf{x}^\mathsf{T} A_r \mathbf{y}$ and Colin's expected payoff will be $\mathbf{y}^\mathsf{T} A_c \mathbf{y}$. Similarly to before, we can then generalise the notion of Nash equilibrium as follows:

Definition 11.3 (Nash Equilibrium for a General 2-player Game)**.** A pair of mixed strategies $\mathbf{x} \in \Delta(R)$ and $\mathbf{y} \in \Delta(C)$ are said to be a *Nash equilibrium* for a two-player game if and only if both:

$$\mathbf{x}^\mathsf{T} A_r \mathbf{y} \geq \mathbf{x}'^\mathsf{T} A_r \mathbf{y} \text{ for all } \mathbf{x}' \in \Delta(R)$$
$$\mathbf{x}^\mathsf{T} A_c \mathbf{y} \geq \mathbf{x}^\mathsf{T} A_c \mathbf{y}' \text{ for all } \mathbf{y}' \in \Delta(C)$$

John Nash showed, as part of his PhD thesis the following general result: *every* game has a Nash equilibrium. The result works even for games that are not zero-sum. It also holds in settings not considered here, such as games with more than 2 players, or games where each player has an infinite number of strategies. However, unlike our proof for the 2-player zero-sum case, his proof relies on results from topology (specifically Brouwer's fixed-point theorem). Moreover, the

proof does not give a method for calculating the equilibrium strategies. Recent results in theoretical computer science suggest that, in contrast to the zero-sum case, computing a Nash equilibrium of a general game might be *impossible* to do efficiently.

11.2 The Prisoner's Dilemma

The most famous example from game theory is the following game called "The Prisoners' Dilemma." Rosemary and Colin have been arrested by the police as suspects. The police question them separately in different rooms. Each can either *snitch* on the other or can *keep quiet*. The payoff matrix looks like this:

	snitch	keep quiet
snitch	$(-2,-2)$	$(0,-3)$
keep quiet	$(-3,0)$	$(-1,-1)$

We can interpret the game as follows. If both keep quiet, the police have only enough evidence to put them in prison for 1 year. If one snitches on the other, then he/she can go free and the other will serve a 3 year sentence. If both snitch, then each of them will have to serve a 2 year sentence. If we examine the payoff matrix, we see that there is only 1 pure Nash equilibrium—that is when both players snitch!

Note that the strange thing about this game is that the players will converge to the outcome that is the *worst* overall in terms of their average happiness. That is, the Nash equilibrium involves 4 total years of prison time, which is more than any other outcome. Additionally, *both* players are better off if they keep quiet. However, because they cannot communicate, neither can ensure that the other player will not snitch.

The prisoner's dilemma has been used to model a variety of scenarios, including nuclear armament in the Cold War and reactions to climate changes. We end the module by returning to our example from the Golden Balls game show. There, we saw that there was a pot of £13,600 available to the players and they had 2 choices—split or steal. We can represent the payoffs to the players in the game show as follows:

	split	steal
split	$(6800,6800)$	$(0,13600)$
steal	$(13600,0)$	$(0,0)$

You can see that this game is very similar to the Prisoner's dilemma. Specifically,

if we identify the players' best responses, we find the following:

	split	steal
split	$(6800, 6800)$	$(\underline{\mathbf{0}}, \underline{\mathbf{13600}})$
steal	$(\underline{\mathbf{13600}}, \underline{\mathbf{0}})$	$(\underline{\mathbf{0}}, \underline{\mathbf{0}})$

Here, there are actually 3 equilibria. However, we could argue that if Rosemary knew that Colin was going to steal, she would probably rather see him get nothing than get £13,600, and similarly for Colin. We can build this extra preference into the players' utility functions by introducing a small value $\epsilon > 0$ representing the satisfaction of seeing the greedy stealing player get nothing. Then, our payoff matrix and best responses look like this:

	split	steal
split	$(6800, 6800)$	$(0, \underline{\mathbf{13600}})$
steal	$(\underline{\mathbf{13600}}, 0)$	$(\underline{\boldsymbol{\epsilon}}, \underline{\boldsymbol{\epsilon}})$

Now, the only Nash equilibrium becomes (steal,steal). Just as in the Prisoner's dilemma, this is the worst overall outcome in terms of total utility for both players. In the episode of Golden balls that we watched, one player (let's use Rosemary in our example) did a strange thing by promising to steal and then give half the money to the other player after the show. Similar scenarios are studied in more advanced game theory, where they are called a cooperative games with side payments or utility sharing.

We can now mathematically show why this was a good strategy. Assuming Rosemary's promise to split the winnings after the show can be trusted, the payoff matrix for the game would become:

	split	steal
split	$(6800, 6800)$	$(0, 13600)$
steal	$(6800, 6800)$	(ϵ, ϵ)

Now, when we compute the best responses for each player, we find:

	split	steal
split	$(\underline{\mathbf{6800}}, 6800)$	$(0, \underline{\mathbf{13600}})$
steal	$(\underline{\mathbf{6800}}, \underline{\mathbf{6800}})$	$(\underline{\boldsymbol{\epsilon}}, \epsilon)$

So, the unique Nash equilibrium is no (steal, split). That is, by offering a payment Rosemary has changed the game so that a mutually beneficial outcome is an equilibrium. This is clearly better for both players than the steal, steal outcome!

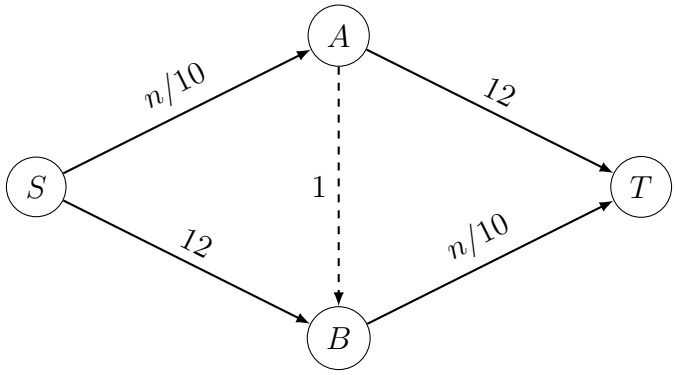


Figure 11.1: A road network with travel times.

11.3 Braess's Paradox

Note: this section is included as an example of a more complicated result from game theory. It is non-examinable (but hopefully interesting!)

In The Prisoner's Dilemma, we have seen an example of a 2-player game in which the only Nash equilibrium in fact leads to a strictly worse outcome for *both* players than another pair of strategies would. Here, we will see another striking example of this phenomenon. Suppose that there are 2 towns S and T joined by a road network, as shown in Figure 12.1. The two arrows $A \to T$ and $S \to B$ represent highways and the two arrows $S \to A$ and $B \to T$ represent smaller roads. We suppose that the average time to travel each highway is a constant: 12 minutes. Due to traffic congestion, the average travel along the two roads depends on the number of users of the road.[1] We suppose that when there are n drivers taking one of the roads the travel time on this road is given by $n/10$.

We can model this as a 100-player game, in which each player is a driver whose available strategies are possible choices of routes from S to T. Each driver wants to select a route with the goal of minimising his or her average travel time. We now show that the unique Nash equilibrium for this setting is when 50 of our 1000 drivers travel along the top road and highway and the other 50 travel along the bottom. Indeed, note that in this case, all drivers have a travel time of $50/10 + 12 = 17$ minutes. Here, the first term is due to the road, which will have 50 drivers, and the second is due to the highway, which has a constant cost of 12 minutes. Consider an arbitrary driver, and let's consider whether it might

[1] Of course, the travel time for the *first* person taking this road will likely not depend on the number of user. Here we imagine a scenario in which 100 drivers commute from S to T every morning at 8:00 AM. On any given day we suppose a driver may arrive slightly before or after any other driver and their relative order is random. Then, for an individual driver it makes sense instead to consider the *average* travel time.

be better to switch to the other route. If the other 99 continue to take the same route, her new average travel time will be $51/10 + 12 = 17.1$ minutes. This is strictly worse. The same is true for any one of our 100 drivers, and so we see that no one has any incentive to change route. In other words, this is indeed a Nash equilibrium.

We now show that no other combination of strategies give a Nash equilibrium. Indeed suppose that x drivers take the top route $S \to A \to T$ and $100 - x$ drivers take the bottom route $S \to B \to T$. Suppose first that $x > 50$. Then, a driver currently on the top route has average travel time $x/10 + 12 > 5 + 12 = 17$. If this driver switched to the bottom route, he would have average travel time $(100-x)/10+12 < (100-50)/10+12 = 17$ and so would do strictly better. Thus, this cannot be a Nash equilibrium. The case in which $x < 50$ can be analysed in exactly the same way. It follows that the *only* Nash equilibrium of our traffic game has exactly $x = 50$ drivers along each route, and the average travel time for each driver is 17 minutes.

Suppose now that a new highway is built between points A and B, as shown with a dashed line in Figure 12.1. The highway has a constant travel time of 1 minutes. Intuitively, building a road should only make the situation better for everyone. However, we will see that, surprisingly, this is not the case!

Suppose we are in the situation where 50 drivers take the top route and 50 take the bottom route. Then, a driver currently taking route $S \to A \to T$ will notice that the route $S \to A \to B \to T$ now takes time $50/10 + 1 + 51/10 = 11.1$ minutes. This is better than 17, so she will switch. It follows that our original Nash equilibrium is no longer an equilibrium after building the road. We now show that in the only equilibrium after building the new road, all 100 drivers travel the path $S \to A \to B \to T$. First, note that, regardless of what the other drivers are doing, no driver ever has an incentive to take either of the highways $S \to B$ or $A \to T$. Indeed, both of these cost 12 minutes, but we could replace these links by $S \to A \to B$ or $A \to B \to T$, respectively, and each of these has a total cost of $n/10 + 1 \leq 11$ (since $n \leq 100$). Thus, it is always better to substitute the route $S \to A \to B$ for $S \to B$ (and similarly $A \to B \to T$ for $B \to T$), and so no Nash equilibrium has any driver travelling along either highway $S \to B$ or $A \to T$. It follows that when *all* 100 drivers take the route $S \to A \to B \to T$ there is no incentive for any of them to change route and so this is a Nash equilibrium. Moreover, if *any* 1 takes another route, we have just seen that this driver would prefer to switch, so no other combination of strategies is a Nash equilibrium.

It follows that the unique Nash equilibrium after we build the extra road $A \to B$ has all 100 drivers travelling the route $S \to A \to B \to T$. The average travel time for a driver on this route is $100/10+1+100/10 = 21$. Notice that this is *larger* than the average travel time before we built the road! This result is called "Braess's

Paradox." Although it indeed seems paradoxical at first, intuitively the reason it happens is that each driver behaves selfishly to minimise their own travel time. Thus, even though we know there is a configuration for everyone that is better (namely 50 drivers taking $S \to A \to T$ and 50 taking $S \to B \to T$), no single driver wants to be the first to switch to this; indeed until enough of the other drivers also switch, this alternative route will costs more. We see that if each driver acts purely selfishly, the overall outcome is worse for everyone than if they together coordinated and agreed not to use the road $A \to B$. This is related to the concept of the "price of anarchy" from algorithmic game theory, which is the worst case ratio between the overall welfare that can be achieved by a centralised planner versus what will happen if all agents act in their own interests.