# IB Optimisation

Martin von Hodenberg (mjv43@cam.ac.uk)

February 2, 2022

These are my notes for the IB course Optimisation, which was lectured in Lent 2022 at Cambridge by Dr V.Jog. These notes are written in LaTeX for my own revision purposes. Any suggestions or feedback is welcome.

## Contents

# §0 Introduction

The course is roughly divided into four parts:

1. Convex optimisation (Lectures 1-3)
2. Lagrange method (Lectures 4-5)
3. Linear programming (Lectures 6-8)
4. Applications of linear programming (Lectures 9-12)

**Definition** (Optimisation problem)

The basic structure of an optimisation problem is as follows:
minimise $f(x)$ for $x \in X$, where in this course we take $X \subset \mathbb{R}^n$. The problem can have 'constraints' $h(x) = b$, where $h : \mathbb{R}^n \to \mathbb{R}^m$. We will look at minimising functions WLOG.

1. $f$ is called the objective function.
2. The components of $x$ are called decision variables.
3. $h(x) = b$ is called a functional constraint.
4. $x \in X$ is called a regional constraint.
5. $\{x : x \in X\}$ is called the feasible set $X(b)$.
6. Problem is feasible if $X(b) \neq \emptyset$, and bounded if the minimum on $X(b)$ is bounded.
7. A point $x^* \in X(b)$ is optimal if it minimises f over $X(b)$. The value $f(x^*)$ is called the optimal set.

# §1 Convex optimisation

## §1.1 Convexity

**Definition**

A set $S \in \mathbb{R}^n$ is **convex** if for all $x, y \in S$, and all $\lambda \in [0,1]$, $(1-\lambda)x + \lambda y \in S$. (The line segment joining $x$ and $y$ lies in $S$.)

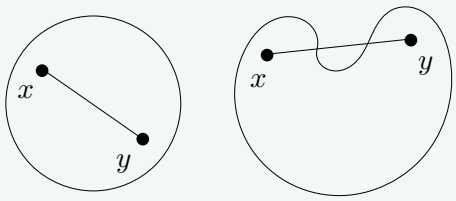



Figure 1: Two subsets of $\mathbb{R}^2$: the left one is convex but the right one is not.

**Definition**

A function $f : S \to \mathbb{R}$ is **convex** if $S$ is convex, and for all $x, y \in S$, and $\lambda \in [0,1]$ , we have

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y).$$

Intuitively, all the tangent lines to the function are below the function.We define $f$ to be concave if $-f$ is convex.

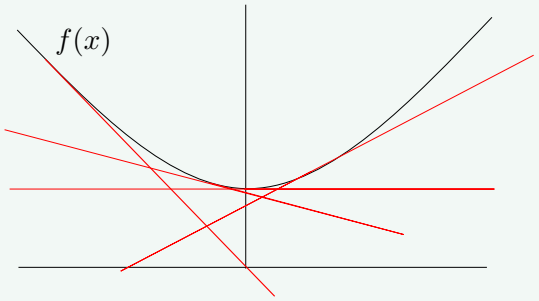



Figure 2: A convex function in $\mathbb{R}^2$.

**Remark.** Linear functions are always both convex and concave, since we get equality in the above equation.

## §1.2 Unconstrained optimisation

We want to find $\min f(x)$ where $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function. This is a particularly simple case of optimisation, since convex functions have the important property that *local information extends globally*. We will now look at a couple of ways to prove convexity, called the first- and second-order conditions.

### §1.2.1 First order conditions for convexity

Intuitively, for a convex function $f : \mathbb{R} \to \mathbb{R}$, we know that all the tangent lines to the function are below the function (see Figure 2). We can express this as

$$f(y) \geq f(x) + (y - x)f'(x).$$

We can generalise this to higher dimensions:

**Theorem 1.2.1** (First-order conditions)
A differentiable $f : \mathbb{R}^n \to \mathbb{R}$ is convex iff for $x, y \in \mathbb{R}^n$ , we have

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

**Remark.** If $\nabla f(x) = 0$, then $f(y) \geq f(x) \implies x$ minimises $f$.

**Proof. Convexity of $f \implies$ F-O conditions hold:**
Let $n = 1$. For $x, y \in \mathbb{R}^n$, $t \in [0, 1]$ we have

$$f((1 - t)x + ty) = f(x + t(y - x)) \leq (1 - t)f(x) + tf(y).$$

Therefore

$$f(y) \geq f(x) + \frac{f(x + t(y - x)) - f(x)}{t}.$$

Taking $t \to 0$ , we conclude

$$f(y) \geq f(x) + f'(x)(y - x).$$

For the general case, define a function $g(t) = f((1 - t)x + ty)$, i.e $g : [0, 1] \to \mathbb{R}$. Since $f$ is convex, $g$ is also convex. We can calculate

$$g'(t) = \nabla f((1 - t)x + ty)^T (y - x).$$

Since $g$ is convex, by the above argument for $n = 1$, we have $g(1) \geq g(0)+g'(0)(1-0)$.

$$\implies f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

**F-O conditions hold $\implies$ convexity:**
Define $x_t = (1 - t)x + ty$. We have the two equations

$$f(x) \geq f(x_t) + \nabla f(x_t)^T (x - x_t) \quad (1)$$
$$f(y) \geq f(x_t) + \nabla f(x_t)^T (y - x_t) \quad (2)$$

Now we add $(1 - t)\times$ equation (1) to $t\times$ equation 2. Noting that $x - x_t = tx - ty$ and $y - x_t = (1 - t)y - (1 - t)x$, we get

$$(1 - t)f(x) + tf(y) \geq f(x_t) + \nabla f(x_t)^T (0).$$

This concludes the proof. □

### §1.2.2 Second-order conditions for convexity

Next we look at second-order conditions. In one dimension, we would expect that $f'$ being increasing, i.e $f''(x) \geq 0$, would give us convexity. We generalise this to $n$ dimensions using the Hessian matrix, which we saw in IA Differential Equations. It has entries $[\nabla^2 f(x)]_{ij} = \frac{\partial^2 f(x)}{\partial x_i x_j}$.

**Definition** (Semidefinite matrix)

A matrix $A$ is called **positive semidefinite** if for all $x \in \mathbb{R}^n$ we have $x^T A x \geq 0$. Equivalently, all eigenvalues of $A$ are non-negative.

**Remark.** The Taylor expansion in $n$ dimensions is

$$f(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x) + \dots.$$

**Theorem 1.2.2** (Second-order condition)

A twice differentiable $f : \mathbb{R}^n \to \mathbb{R}$ is convex if $\nabla^2 f(x)$ is positive semidefinite, i.e $x^T \nabla^2 f(z) x \geq 0$ for all $x, z \in \mathbb{R}^n$.

**Proof.** Using the Taylor expansion of $f$ we have

$$f(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x).$$

where $z = (1 - \lambda)x + \lambda y$ for some $\lambda \in [0, 1]$. (IA Analysis).
We have $f(y) \geq f(x) + \nabla f(x)^T (y - x)$. Then by the first-order condition theorem, $f$ is convex. □

## §1.3 The gradient descent algorithm

Recall we have an unconstrained optimisation problem:

$$\text{minimise } f(x), f : \mathbb{R}^n \to \mathbb{R} \text{ is a convex function.}$$

In order to solve this, we could apply agree a 'greedy method' where we pick an initial point $x_0$ and iteratively pick points $x_1, x_2, \dots$ near this point so that the gradient of $f$ at $x_n$ is decreasing. In which direction should we decrease $x_n$? Note that by Taylor's expansion, with small $\epsilon > 0$,

$$f(x - \epsilon \nabla f(x)) \approx f(x) - \epsilon \nabla f(x)^T \dot{\nabla} f(x) = f(x) - \epsilon ||f(x)||^2 \leq f(x).$$

We call the direction $-\nabla f(x)$ the descending direction. (Note we could also choose any other direction $v$ so that $f(x)^T \dot{v} < 0$.) We now formalise this process in an algorithm.

**Definition** (Gradient descent method)

Define an algorithm:

1. Start at some point $x_0$. Set $t = 0$.
2. Repeat the following:

- Find a descending direction $v_t$ (e.g $-\nabla f(x_t)$)
- Choose a step size $\eta_t$ (can depend on $t$)
- Update $x_{t+1} = x_t + \eta_t v_t$

3. Stop when some criterion is met (e.g $\nabla f(x_t) = 0$, $t$ is large enough)

For the scope of this course, we need to make assumptions about our convex function.

### §1.3.1 Smoothness assumption

**Definition** (Smoothness assumption)

We say that a continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ is $\beta$-smooth or $\beta$-Lipschitz if

$$||\nabla f(x) - \nabla f(y)|| \leq \beta ||x - y||.$$

Those who have taken IB Analysis and Topology will be familiar with this concept.

**Remark.** If $f$ is twice-differentiable, then $\beta$-smoothness implies that $\nabla^2 f(x) \leq \beta I$. Equivalently, all eigenvalues of $\nabla^2 f(x)$ are $\leq \beta$. $\beta$-smoothness can be summed up by the fact that *the linear approximation is close to $f$ in a small neighbourhood around $x$.*

**Proposition 1.3.1**

If $f$ is $\beta$-smooth and convex, then

$$f(x) + \nabla f(x)^T (y - x) \leq f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{\beta}{2} ||x - y||^2.$$

**Proof.** The left-hand inequality follows by convexity.
For the right hand inequality, by Taylor's theorem,

$$\begin{aligned} f(y) =& \nabla f(x)^T (y - x) + \frac{1}{2} (y - x)^T \nabla^2 f(z)(y - x) \\ &\leq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} (y - x)^T (\beta I)(y - x) \\ &= f(x) + \nabla f(x)^T (y - x) + \frac{\beta}{2} ||x - y||^2 \end{aligned}$$

□

**Corollary 1.3.2**

At any point $x$,

$$f(x - \frac{1}{\beta} \nabla f(x)) \leq f(x) - \frac{1}{2\beta} ||f(x)||^2.$$

**Proof.** Let's look at

$$f(x) + \nabla f(x)^T (y - x) + \frac{\beta}{2} ||x - y||^2,$$

and try to minimise it over $y$ for a fixed $x$.

$$\nabla y(f(x)^T(y-x)+\frac{\beta}{2}||x-y||^2) = \nabla f(x) - \beta(x-y) = 0.$$

$$\implies \nabla \frac{\nabla f(x)}{\beta} = x - y \implies y = x - \frac{1}{\beta}\nabla f(x).$$

Plug this value of $y$ into the above claim:

$$\begin{aligned} f(x-\frac{1}{\beta}\nabla f(x)) &\leq f(x) + \nabla f(x)^T(\frac{-1}{\beta}\nabla f(x)) + \frac{\beta}{2}||\frac{1}{\beta}\nabla f(x)||^2 \\ &= f(x) - \frac{1}{\beta}||\nabla f(x)||^2 + \frac{1}{2\beta}||\nabla f(x)||^2 \\ &= f(x) - \frac{1}{2\beta}||\nabla f(x)||^2 \end{aligned}$$

□

**Corollary 1.3.3** (Improved first-order condition)

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2\beta}||\nabla f(x) - \nabla f(y)||^2.$$

**Proof.** For any $z$, we have

$$f(x) + \nabla f(x)^T(z-x) \leq f(z) \leq f(y) + \nabla f(y)^T(z-y) + \frac{\beta}{2}||z-y||^2.$$

This implies

$$f(x) - f(y) \leq \nabla f(x)^T(x-z) + \nabla f(y)^T(z-y) + \frac{\beta}{2}||z-y||^2.$$

To minimise the RHS, set $\nabla_z = 0$. We get $-\nabla f(x) + \nabla f(y) + \beta(z-y) = 0$, which implies

$$z = \frac{f(x) - f(y)}{\beta} + y.$$

Subbing into the RHS, we get

$$f(x) - f(y) \leq \nabla f(x)^T(x-y) - \frac{1}{2\beta}||\nabla f(x) - \nabla f(y)||^2.$$

□

### §1.3.2 Strong convexity assumption

We now introduce a new type of convexity we assume. In essence, it tells us that *if the gradient is small, we are close to the optimum.*

**Definition** (Strong convexity assumption)

A function $f : \mathbb{R}^n \to \mathbb{R}$ is $\alpha$-strongly convex if

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\alpha}{2}||x - y||^2.$$

If $f$ is twice differentiable, then

$$\nabla^2 f(x) \geq \alpha I \ \forall x.$$

**Proposition 1.3.4**

Let $f$ be $\alpha$-strongly convex. Let $p^*$ be the optimal cost. Then for any $x$ we have

$$p^* \geq f(x) - \frac{1}{2\alpha}||\nabla f(x)||^2.$$

**Remark.** If $||\nabla f(x)|| \leq \sqrt{2\alpha\epsilon}$ then $p^* \geq f(x) - \epsilon$, i.e

$$p^* \leq f(x) \leq p^* + \epsilon.$$

**Proof.** The $\alpha$-strong convexity assumption gives us

$$\min_y f(y) \geq \min_y f(x) + \nabla f(x)^T(y - x) + \frac{\alpha}{2}||x - y||^2.$$

Setting $\nabla_y$ of the RHS=0, we get $y = x - \frac{\nabla f(x)}{\alpha}$.
Plugging this value of $y$ in the RHS, we get

$$f(x) + \nabla f(x)^T(\frac{-\nabla f(x)}{\alpha}) + \frac{\alpha}{2}||\frac{\nabla f(x)}{\alpha}||^2 = f(x) - \frac{1}{2\alpha}||\nabla f(x)||^2.$$

□

We might also want to ask: how close is the true optimiser $x^*$ to our current point $x$? It turns out $\alpha$-strong convexity gives us an answer to this too.

**Proposition 1.3.5**

Let $x \in S$, and $x^*$ be the true optimiser. Then

$$||x - x^*|| \leq \frac{2}{\alpha}||\nabla f(x)||.$$

**Proof.**

$$\begin{aligned} f(x^*) &\geq f(x) + \nabla f(x)(x^* - x) + \frac{\alpha}{2}||x - x^*||^2 \\ &\geq f(x) - ||\nabla f(x)||||x^* - x|| + \frac{\alpha}{2}||x - x^*||^2 \text{ by Cauchy-Schwartz} \end{aligned}$$

We already know that $f(x^*) \leq f(x)$. Therefore

$$0 \geq f(x^*) - f(x) \geq -||\nabla f(x)||\ ||x^* - x|| + \frac{\alpha}{2}||x - x^*||^2.$$

So $||\nabla f(x)||\ ||x - x^*|| \leq \frac{2}{\alpha}||\nabla f(x)||^2$, and thus

$$||x - x^*|| \leq \frac{2}{\alpha}||\nabla f(x)||.$$

□

### §1.3.3 Efficiency of gradient descent

**Theorem 1.3.6**

Gradient descent with step size $\frac{1}{\beta}$ satisfies:

$$f(x_t) - f(x^*) \leq (1 - \frac{\alpha}{\beta})^t (f(x_0) - f(x^*)) \leq e^{\frac{-\alpha t}{\beta}} (f(x_0) - f(x^*)) \leq e^{\frac{-\alpha t}{\beta}} \frac{\beta}{2}||x^* - x_0||^2.$$

Note this is very efficient; the error decreases exponentially!

**Proof.**

$$\begin{aligned} f(x_{t+1}) - f(x*) &\leq f(x_t) - f(x^*) - \frac{1}{2\beta}||\nabla f(x_t||^2 \\ &\leq f(x_t) - f(x^*) - \frac{\alpha}{\beta}(f(x_t) - f(x^*)) \text{ by strong convexity} \\ &\leq (1 - \frac{\alpha}{\beta})(f(x_t) - f(x^*)) \end{aligned}$$

Using induction, conclude that

$$f(x_t) - f(x^*) \leq (1 - \frac{\alpha}{\beta})^t (f(x_0) - f(x^*)).$$

Because $f$ is $\beta$-smooth, we have

$$f(x_0) \leq f(x^*) + \underbrace{\nabla f(x)^t (x_0 - x^*)}_{=0} + \frac{\beta}{2}||x_0 - x^*||^2.$$

□

**Remark.** With this algorithm, the number of steps needed to ensure error $\leq \epsilon$ is

$$\frac{\beta}{\alpha} \log(\frac{f(x_0) - f(x^*)}{\epsilon}).$$

The $\log(\frac{1}{\epsilon})$ dependence is called 'linear convergence'.

## §1.4 Alternatives to gradient descent

### §1.4.1 Newton's method

When we use gradient descent, the factor $(1 - \frac{\alpha}{\beta})$ controls the speed of convergence. We call $\frac{\beta}{\alpha}$ the **condition number** of $f$ (note we always have $\beta > \alpha$). If the condition number is large, then convergence will be slow and we may want to consider other algorithms. Let's look at an example:

**Example**

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be given by

$$f(x, y) = \frac{1}{2}(x^2 + 100y^2).$$

Now we have

$$\nabla^2 f(x, y) = \begin{pmatrix} 1 & 0 \\ 0 & 100 \end{pmatrix}.$$

Note that $I \leq \nabla^2 f(x, y) \leq 100I$. Our condition number is 100, which is very large! The contours of $f$ are very stretched ellipses in $\mathbb{R}^2$. If we use gradient descent on a point, it will comparatively overshoot a very large amount in the $y$ direction. Therefore convergence will be slow.

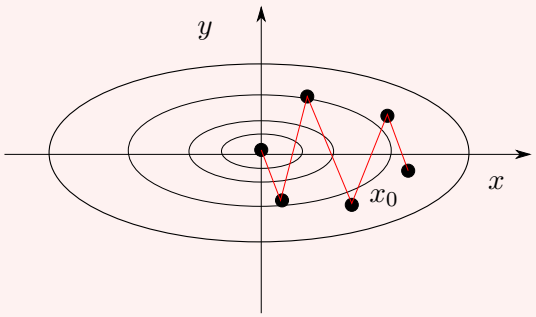



Figure 3: The path oscillates as it overshoots each time in the $y$ direction.

We need another method in these cases. This is given by Newton's method:

**Definition** (Newton's method)

Here we use a modified version of gradient descent, with

$$x_{t+1} = x_t - (\nabla^2 f(x))^{-1} \nabla f(x_t).$$

The derivation of this comes from the second order Taylor approximation of $f$ at $x_t$:

$$f(x) \approx f(x_t)^T (x - x_t) + \frac{1}{2}(x - x_t)^T \nabla^2 f(x_t)(x - x_t).$$

If we minimise the RHS wrt. $x$, then we get $x = x_t - (\nabla^2 f(x))^{-1} \nabla f(x)$. This method of minimising the second-order approximation is shown in the figure below:

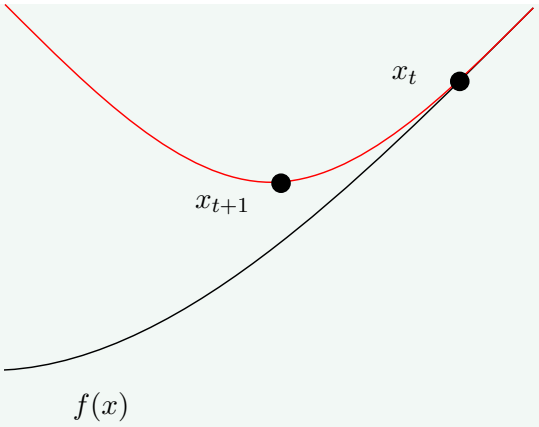



Figure 4: The second-order approximation, shown in red, is repeatedly minimised as we increment $t$.

**Remark.** Intuitively, Newton's method is excellent for functions with accurate second-order approximations.

### §1.4.2 Convergence of Newton's method

We will not do the analysis in this course, but Newton's method satisfies

$$||x_{t+1} - x^*|| \leq C||x_t - x^*||^2, \text{ as long as } x_t \text{ is close enough to } x^*.$$

In one dimension, Newton's method is a root-finding algorithm:
Take $f : \mathbb{R} \to \mathbb{R}$, and let $f' = g$. Write

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} = x_t - \frac{g(x_t)}{g'(x_t)}.$$

If we set $f'(x_{t+1}) = g(x_{t+1}) = 0$, i.e looking for a stationary point of $f$ (which is the same as a root of $g$ if $g$ is convex) then we can write $g(x_{t+1}) \approx g(x_t) + g'(x_t)(x_{t+1} - x_t) = 0$, and this gives the previous equation as expected. The root-finding algorithm is illustrated below:

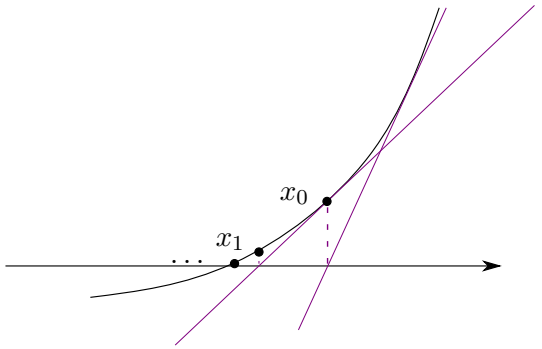



Figure 5: Newton's method applied in one dimension as a root-finding algorithm. Note the speed of convergence displayed.

### §1.4.3 Barrier methods

What if the function we had to minimise involved constraints? We can write the optimisation problem as

$$\text{minimise } f(x) \text{subject to} f_i(x) \leq 0 \ \forall 1 \leq i \leq n.$$

The way we approach this is to transform the constrained problem into an unconstrained one. The way we do this is by defining some new functions for the new optimisation problem:

$$\text{minimise } f(x) + \sum_{i=1}^{n} \phi(f_i(x)), \phi = \begin{cases} \infty & f_i(x) > 0 \text{ for any } 1 \leq i \leq m \\ 0 & f_i(x) \leq 0 \text{ for all } 1 \leq i \leq m \end{cases}$$

This may still be difficult to deal with, so we can define a **logarithmic barrier function**:

$$\text{minimise } tf(x) - \sum \log(-f_i(x)).$$

Here we are taking $\phi(x) = -\log(-x)$, so we are using an approximation: however we can make this negligible in practice by choosing suitable $t$.

**Definition** (Barrier method)

We define an algorithm:

1. Find a point $x$ in the feasible set. Set $t > 0$.
2. Repeat:
   - Solve $\min tf(x) + \sum_{i=1}^{n} \phi(f_i(x))$ with $x$ as the starting point, use Newton's method.
   - Set $x = x^*(t)$.
   - Stop if $t$ is large enough
   - Increase $t = \alpha t$ for $\alpha > 1$.

# §2 The Lagrange multiplier method

Suppose we want to minimise $f(x)$ for $x \in X$ subject to constraints $h(x) = b$ where $h : \mathbb{R}^n \to \mathbb{R}^m$.
The **Lagrangian** associated with this problem is

$$L(x, \lambda) = f(x) - \lambda^T (h(x) - b).$$

Note that $\lambda \in \mathbb{R}^m$. We want to minimise $L(x, \lambda)$ over $x \in X$ for some value of $\lambda$.

## §2.1 Lagrange sufficiency

**Theorem 2.1.1** (Lagrange sufficiency theorem)

Suppose we can find a $\lambda^*$ such that:

1. $\min_{x \in X} L(x, \lambda^*) = L(x^*, \lambda^*)$ for some $x^* \in X$.
2. $x^* \in X(b) = \{x : x \in X, h(x) = b\}$, the feasible set for our constrained

problem.

Then $x^*$ is optimal for 1, i.e $\min_{x\in X(b)} f(x) = f(x*)$.

**Proof.** Condition 2 says that $f(x^*) \geq \min_{x\in X(b)} f(x)$ because $x^*$ is feasible. We have

$$\begin{aligned}\min_{x\in X(b)} f(x) &= \min_{x\in X(b)} f(x) - (\lambda^*)^T(h(x)-b)\\ &\geq \min_{x\in X} f(x) - (\lambda^*)^T(h(x)-b)\\ &= L(x^*,\lambda^*)\\ &= f(x^*) - (\lambda^*)^T(h(x^*)-b)\\ &= f(x^*).\end{aligned}$$

□

Let's look at an example.

**Example**

Minimise $-x_1 - x_2 + x_3$ subject to $x_1^2 + x_2^2 = 4, x_1 + x_2 + x_3 = 1$. Here we have

$$h(x_1,x_2,x_3) = \begin{pmatrix} x_1^2 + x_2^2 \\ x_1 + x_2 + x_3 \end{pmatrix}, b = \begin{pmatrix} 4 \\ 1 \end{pmatrix}.$$

Our Lagrangian is

$$L(x,\lambda) = (-x_1 - x_2 + x_3) - \lambda_1(x_1^2 + x_2^2 - 4) - \lambda_2(x_1 + x_2 + x_3 - 1).$$

We can rewrite this as

$$L(x,\lambda) = (-(1+\lambda_2)x_1 - \lambda_1 x_1^2) = (-(1+\lambda_2)x_2 - \lambda_1 x_2^2) + (1-\lambda_2)x_3 + (4\lambda_1 + \lambda_2).$$

Now fix $\lambda$, and try to find $\min_{x\in\mathbb{R}^n} L(x,\lambda)$. We'll only be interested in $(\lambda_1,\lambda_2)$ so that the minimum is finite.

- If $\lambda_1 > 0$, the infinimum is $-\infty$.
- If $\lambda_2 \neq 1$, the infinimum is $-\infty$.

Let's look at $\lambda_1 \leq 0, \lambda_2 = 1$.

$$\frac{d}{dx_1}(-(1+\lambda_2)x_1 - \lambda_1 x_1^2) = -(1+\lambda_2) - 2\lambda_1 x_1 = 0 \implies x_1 = \frac{-1}{\lambda_1}.$$

Similarly, differentiating wrt. $x_2$ gives $x_2 = \frac{-1}{\lambda_1}$. Can we pick $\lambda_1$ so that $(x_1,x_2)$ satisfy the constraints? We need $x_1^2 + x_2^2 = 4, x_1 + x_2 + x_3 = 1$. Therefore

$$x_1^2 = x_2^2 = 2 \implies x_1 = x_2 = \sqrt{2}.$$

Since $\lambda_1 \leq 0$, our optimiser is

$$x = (\sqrt{2}, \sqrt{2}, 1 - 2\sqrt{2}).$$

Let's formalize the above method for solving

$$\min_{x\in X} f(x) \text{ with } h(x) \leq b.$$

1. Add a slack variable $s$ to transform the problem into

$$\min_{x\in X} f(x) \text{ with } h(x) + s = b, s \geq 0.$$

2. Set $L(x, \lambda, s) = f(x) - \lambda^T(h(x) + s - b)$, and let

$$\Lambda = \{\lambda : \inf_{x\in X, x\geq 0} L(x, \lambda, s) \geq -\infty\}.$$

3. For each $\lambda \in \Lambda$, find $x^*(\lambda)$,$s^*(\lambda)$ such that

$$\min_{x\in X, s\geq 0} L(x, \lambda, s) = L(x^*(\lambda), \lambda, s^*(\lambda)).$$

4. Find a $\lambda^* \in \Lambda$ such that $(x^*(\lambda^*), s^*(\lambda^*))$ is feasible; i.e $h(x^*(\lambda^*)) = b$ and $s^*(\lambda^*) \geq 0$.

## §2.2 Complementary slackness

Let's suppose we want to minimise $f(x) - \lambda^T(h(x) + s - b)$ subject to $x \in X, s \geq 0$. Suppose for $\lambda \in \Lambda$, we solve to arrive at $x^*(\lambda), s^*(\lambda)$. Note we must have all the components $\lambda_i \leq 0$ in order to keep $\lambda \in \Lambda$. We have

$$-\lambda^T s = -\sum \lambda_i s_i = \sum_{i=1}^{m} (\lambda_i)\dot{s}_i.$$

$\min -\lambda^T s = 0$, which means $\lambda_i s_i = 0 \; \forall 1 \leq i \leq m$. Consider our inequalities $h_i(x) + s_i = b_i$. If any of these inequalities are not equalities, then for that $i$ we must have $\lambda_i = 0$. This is called **complementary slackness**.

**Example**

Problem:

$$\min x_1 - 3x_2 \text{ subject to } x_1^2 + x_2^2 \leq 4, x_1 + x_2 \leq 2.$$

Let's apply our algorithm.

1. Add slack variables: our problem is now

$$\min x_1 - 3x_2 \text{ subject to } x_1^2 + x_2^2 + s_1 = 4, x_1 + x_2 + s_2 = 2, s_1, s_2 \geq 0.$$

2. Work out

$$L(x, \lambda, s) = ((1-\lambda_2)x_1 - \lambda_1 x_1^2) + ((-3-\lambda_2)x_2 - \lambda_1 x_2^2) - \lambda_1 s_1 - \lambda_2 s_2 + (4\lambda_1 + 2\lambda_2).$$

3. We must have $\lambda_1, \lambda_2 \leq 0$ by complementary slackness. At the optimum, $\lambda_1 s_1 = \lambda_2 s_2 = 0$. By differentiating $L$, we get the system of equations

$$(1 - \lambda_2) - 2\lambda_1 x_1 = 0.$$
$$-3 - \lambda_2 - 2\lambda_1 x_2 = 0.$$

These show we can't have $\lambda_1 = 0$ and thus $\lambda_1 < 0$ and $s_1 = 0$. There are two more cases:

- If $\lambda_2 < 0$, then $s_2 = 0$. We have the two equations above, and then by complementary slackness we also know that

$$x_1^2 + x_2^2 = 4, x_1 + x_2 = 2.$$

  Solving these gives $(x_1, x_2) = (0, 2)$ or $(2, 0)$. But with both of these solutions the first two equations give one of the $\lambda_i > 0$. So we don't have any solutions in this case.

- The last case is $\lambda_1 < 0, \lambda_2 = 0, s_1 = 0$. Here we get the two equations above and $x_1^2 + x_2^2 = 4$. Here we get

$$x_1 = \frac{1}{2\lambda_1}, x_2 = \frac{-3}{2\lambda_1}.$$

  Therefore $\lambda_1^2 = \frac{5}{8}$ and thus

$$(x_1, x_2) = (-\sqrt{\frac{2}{5}}, 3\sqrt{\frac{2}{5}}).$$

  So we are done by Lagrange sufficiency.

## §2.3 Strong duality and shadow prices

For some optimisation problems that we solve using the Lagrange method, for *every* $\lambda$ we can find a feasible $x^*$ such that

$$\min_{x \in X} L(x, \lambda) = L(x^*, \lambda).$$

This property of an optimisation problem is called **strong duality**. Before we do this, however, let's start with a weaker theorem.

**Theorem 2.3.1** (Weak duality)
For a Lagrange problem define

$$g(\lambda) = \inf_{x \in X} L(x, \lambda).$$

If $x \in X(b)$, and $\lambda \in \Lambda$, then $f(x) \geq g(x)$. In particular,

$$\inf_{x \in X(b)} f(x) \geq \sup_{\lambda \in \Lambda} g(\lambda).$$

**Proof.** For all $\lambda \in \Lambda$,

$$\begin{aligned}\inf_{x \in X(b)} f(x) &= \inf_{x \in X(b)} (f(x) - \lambda^T (h(x) - b)) \\ &\geq \inf_{x \in X} (f(x) - \lambda^T (h(x) - b)) \\ &= \inf_{x \in X} L(x, \lambda) \\ &= g(\lambda).\end{aligned}$$

□

The problem of maximising $g(\lambda)$ subject to $\lambda \in \Lambda$ is called the **dual problem**, and the problem of minimising $f(\lambda)$ subject to $x \in X(b)$ is called the **primal problem**.
The **duality gap** is

$$\inf_{x \in X(b)} f(x) - \sup_{\lambda \in \Lambda} g(\lambda).$$

If the gap is 0, we say that **strong duality** holds.

**Definition** (Hyperplane)

A function $\phi : \mathbb{R}^m \to \mathbb{R}$ is said to have a supporting hyperplane at $b$ if there exists a $\lambda \in \mathbb{R}^m$ such that for all $c \in \mathbb{R}^m$, the following holds:

$$\phi(c) \geq \phi(b) + \lambda^T (c - b).$$

This is easy to visualise in one dimension: There exists a tangent to the curve $\phi$ at $(b, \phi(b))$ such that the tangent is below the curve.

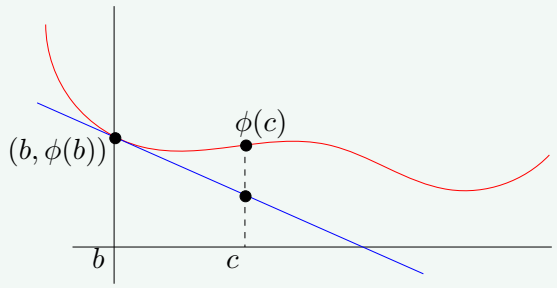



Figure 6: The hyperplane in one dimension is a tangent below the curve with slope $\lambda$.

**Definition** (Value function)

Define the **value function** $\phi : \mathbb{R}^m \to \mathbb{R}$ associated with the primal problem as follows:

$$\phi(c) = \inf_{x \in X(c)} f(x).$$

**Theorem 2.3.2** (Strong duality)

Strong duality holds if and only if the vlaue function $\phi$ has a separating hyperplane at $b$.

**Proof.** We prove both directions.
**Separating hyperplane $\implies$ strong duality:**
There exists $\lambda$ such that $\phi(c) \geq \phi(b) + \lambda^T(c-b)$. Then we have

$$\begin{aligned} g(\lambda) &= \inf_{x \in X}(f(x) - \lambda^T(h(x) - b)) \\ &= \inf_c \inf_{x \in X}(f(x) - \lambda^T(h(x) - c) - \lambda^T(c-b)) \\ &= \inf_c(\phi(c) - \lambda^T(c-b)) \\ &\geq \phi(b). \end{aligned}$$

Weak duality gives $g(\lambda) \leq \phi(b)$, and so $g(\lambda) = \phi(b)$; i.e, strong duality holds.
**Strong duality holds $\implies$ separating hyperplane exists at $b$:**
There exists $\lambda$ such that $g(\lambda) = \phi(b)$.

$$\begin{aligned} \phi(b) = g(\lambda) &= inf_{x \in X}(f(x) - \lambda^T(h(x) - b)) \\ &= inf_{x \in X}(f(x) - \lambda^T(h(x) - c) - \lambda^T(c-b)) \\ &\leq \phi(c) - \lambda^T(c-b) \text{ by weak duality} \end{aligned}$$

This means that $\lambda$ gives the supporting hyperplane at $b$. □

### §2.3.1 When does $\phi(b)$ have a separating hyperplane?

**Theorem 2.3.3** (Separating hyperplanes and convex functions)
A function $\phi : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if every point $b \in \mathbb{R}^n$ has a separating hyperplane.

**Proof.** We state this theorem without proof. □

**Theorem 2.3.4** (Convexity conditions)
Consider the problem of minimising $f(x)$ with $x \in X, h(x) \leq b$. Then $\phi$ is convex if:

1. $X$ is convex
2. $f$ is convex
3. $h$ is convex

**Proof.** This is left to be proved on Example Sheet 1. □

### §2.3.2 Economics interpretations

Let's say we have a factory owner. He makes $n$ types of products using $m$ types of raw materials (which have a finite supply). Let's say that with $x = (x_1, \ldots, x_m)$ raw materials he makes a profit of $f(x)$.
Further, let $h_j(x)$ is the raw material consumed of type $j$. Our problem is to maximise $f(x)$ such that $h_i(x) \leq b_i$ for $1 \leq i \leq m$, and $x \geq 0$.

If $(\epsilon_1, \dots, \epsilon_m)$ amount of raw material is offered to the factory owner, how much is it worth? For small enough $\epsilon$, we have

$$\phi(b+\epsilon) - \phi(b) \approx \sum_{j=1}^{m} \frac{\partial \phi(b)}{\partial b_j} \epsilon_j.$$

The vector of prices $(\frac{\partial \phi(b)}{\partial b_1}, \dots, \frac{\partial \phi(b)}{\partial b_m}) = \nabla \phi(b)$. These are called "shadow prices".

**Theorem 2.3.5**

If $\phi$ is differentiable at $b$ and has a separating hyperplane given by $\lambda$, then $\lambda = \nabla \phi(b)$.

**Proof.** Let $a = (a_1, \dots, a_m)$ be an arbitrary vector. Pick $\delta > 0$.

$$\begin{aligned} &\frac{\phi(b+\delta a) - \phi(b)}{\delta} \geq \lambda^T a \\ \implies &\nabla \phi(b)^T a \geq \lambda^T a \text{ for every vector } a. \\ \implies &\nabla \phi(b) = \lambda. \end{aligned}$$

□

The Lagrange multiplier at $b$ is the vector of partial derivatives, i.e the vector of shadow prices. The intutive interpretation of 'not wanting to pay any more' when you have enough of a certain raw material (that shadow price being 0) is an interpretation of complementary slackness.

Now let's look at the dual problem. If the raw material seller charges a price $\lambda$, and the finished product is consumed when he sells it, then

$$\lambda^T (h(x) - b) - f(x)$$

is the Lagrangian he is trying to minimise. But the factory producer is producing $x^*(\lambda)$ to maximise

$$f(x) - \lambda^T (h(x) - b).$$

Let's call $g(\lambda) = \lambda^T (h(x^*(\lambda) - b) - f(x^*(\lambda)))$. The seller is trying to maximise $g$. This provides a good example of the dual problem.

# §3 Solving linear programs

**Definition** (Linear program)

A general linear program is an optimisation problem of

$$\text{minimise } c^T x, \text{ subject to } \begin{cases} a_i^T x \geq b_i & i \in M_1 \\ a_i^T x \leq b_i & i \in M_2 \\ a_i^T x = b_i & i \in M_3 \\ x_j \geq 0 & j \in N_1 \\ x_j \geq 0 & j \in N_2 \end{cases}$$

Two linear programs are equivalent if given a feasible solution for one formulation, we can construct a feasible solution for the other formulation with the same cost. The **general form** of a linear program is:

$$\text{minimise } c^T x \text{ subject to } Ax \geq b.$$

where

$$A = \begin{pmatrix} \dots & A_1^T & \dots \\ \dots & A_2^T & \dots \\ & \vdots & \\ \dots & A_m^T & \dots \end{pmatrix}, \; b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

A linear problem is said to be in **standard form** if it has the form

$$\text{minimise } c^T x \text{ subject to } Ax \geq b, x \geq 0.$$

We can do this WLOG since we can split $x$ into two vectors $x^+ - x^-$ such that each of their entries are positive and then minimise $c^T(x^+ - x^-)$ with the respective modified version of $A$.

## §3.1 Maximising convex functions

Recall that $\min c^T x = \max -c^T x$. So solving linear programs is a special case of maximising convex functions. Let's consider the general problem for a convex set $C$:

$$\text{maximise } f(x) \text{ subject to } x \in C.$$

Take $z = (1-\lambda)x + \lambda y$ where $x, y \in C$. We have

$$f(z) \leq (1-\lambda)f(x) + \lambda f(y) \leq \max\{f(x), f(y)\}.$$

This is true for all $x, y \in C$, so the global maximum must be maximised somewhere on the 'boundary' of $C$.

**Definition** (Extreme point)

A point $x$ in a convex set $C$ is an **extreme point** if it cannot be written as $(1-\delta)y + \delta z$ for $\delta \in (0,1)$ and $y, z$ distinct.

**Remark.** "Convex functions on convex sets are maximised at extreme points".

## §3.2 Basic solutions and basic feasible solutions

Recall our linear problem:
minimise $c^T x$ such that $Ax = b, x \geq 0$ with $A \in L(\mathbb{R}^n, \mathbb{R}^m), x \in \mathbb{R}^n, b \in \mathbb{R}^m$.
We make two assumptions:

$(*)$ All $m$ rows of $A$, $(a_1^T, \dots, a_m^T)$ are linearly independent.

$(**)$ Every set of $m$ columns of $A$ is linearly independent WLOG. We can see these first two assumptions as getting rid of redundant variables.

$(***)$ Every basic solution has exactly $m$ non-zero entries. (Non-degeneracy assumption)

We can't make assumption $(***)$ WLOG, but in this course we will only consider linear problems that satisfy this assumption. What's a basic solution, you may ask? We'll define it now:

**Definition** (Basic solution)

A point $x$ is a basic solution if it satisfies $Ax = b$, and $x$ has at most $m$ non-zero entries. (Recall $x \in \mathbb{R}^n$).

**Definition** (Basic feasible solution)

$x$ is a **basic feasible solution** (BFS) if $x$ is a basic solution and $x \geq 0$.

### §3.2.1 How to find basic solutions

Pick $B_{(1)}, \ldots, B_{(m)}$ to be the indices where $x$ is non-zero. So

$$\begin{pmatrix} \vdots & \vdots & & \vdots \\ A_1 & A_2 & \ldots & A_n \\ \vdots & \vdots & & \vdots \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ x_{B_{(1)}} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ A_{B_{(1)}} & A_{B_{(2)}} & \ldots & A_{B_{(m)}} \\ \vdots & \vdots & & \vdots \end{pmatrix} \begin{pmatrix} x_{B_{(1)}} \\ \vdots \\ x_{B_{(m)}} \end{pmatrix} = B \begin{pmatrix} x_{B_{(1)}} \\ \vdots \\ x_{B_{(m)}} \end{pmatrix} = b.$$

Therefore for this matrix $B$,

$$\begin{pmatrix} x_{B_{(1)}} \\ \vdots \\ x_{B_{(m)}} \end{pmatrix} = B^{-1} b.$$

**Definition** (Basis matrix)

The matrix $B$ we saw is called the **basis matrix**. Its columns are the **basis columns**, and $(x_{B_{(1)}}, \ldots, x_{B_{(m)}})$ are the **basis variables**.

## §3.3 Extreme points of the feasible set

**Theorem 3.3.1** (Extreme points are basic feasible solutions)

A vector $x$ is an extreme point of the set $\{x : \ Ax = b, x \geq 0\}$ if and only if $x$ is a basic feasible solution.

**Remark.** Linear programs are optimised at extreme points, and the above theorem says extreme points are basic feasible solutions. So it's enough to look at basic feasible solutions (which we know how to do!). Therefore, we could develop an algorithm:

1. Pick all possible $m$ columns of $A$. There are $\binom{n}{m}$ such choices.
2. Find the basic solution, and check if it is a BFS.
3. If it is a BFS, calculate $c^T x$, store the $x$ which has the least cost.

**Proof.** We prove both directions.
**BFS $\implies$ extreme point:**
Suppose $x$ is a BFS, and suppose $\exists\, y, z$ such that $x = (1-\delta)y + \delta z$, where $\delta \in (0,1)$. Remember $x \in \mathbb{R}^n$ and $x \geq 0$, and has $m$ non-zero entries. However, $y, z$ are also feasible, and therefore $y, z \geq 0$. So

$$y_j, z_j = 0 \text{ for } j \notin \{B_{(1)}, \ldots, B_{(m)}\}.$$

Define

$$y_B = \begin{pmatrix} y_{B_{(1)}} \\ \vdots \\ y_{B_{(m)}} \end{pmatrix}, z_B = \begin{pmatrix} z_{B_{(1)}} \\ \vdots \\ z_{B_{(m)}} \end{pmatrix}, .$$

We know $By_B = b$ and $Bz_B = b$ because $Ay = Az = b$. Therefore $y_B = z_B = B^{-1}$, and so $x = y = z$. Thus $x$ is an extreme point.
**Not BFS $\implies$ not extreme point:**
Pick $x$ which is not a BFS. Let the indices where $x$ is non-zero be $i_1, i_2, \ldots, i_r$ where $r > m$. The columns $A_{i_1}, \ldots, A_{i_r}$ are linearly dependent since $r > m$. So we can pick some $w_{i_1}, \ldots, w_{i_r}$ so that

$$\sum_{k=1}^{r} w_{i_k} A_{i_k} = 0.$$

Define the components of the column vector $w$ by

$$w_i = \begin{cases} 0 & i \notin \{i_1, \ldots, i_r\} \\ w_{i_k} & i = i_k \end{cases}.$$

It's easy to check that $Aw = 0$. Now consider the two points $x + \epsilon w$ and $x - \epsilon w$. Note both $A(x + \epsilon w) = 0$, $A(x - \epsilon w) = 0$. We can find an $\epsilon > 0$ so that $x + \epsilon w \geq 0, x + \epsilon w \geq 0$. This means

$$x = \frac{1}{2}(x + \epsilon w) + \frac{1}{2}(x - \epsilon w) = 0.$$

and thus is not an extreme point.

□

## §3.4 Duality in linear programming

**Theorem 3.4.1** (Strong duality for linear programs)

If a linear program is bounded and feasible, then it satisfies strong duality.

**Proof.** We have that $X$ is a convex set, $f$ is convex, and that the boundary functions $h_i$ are convex for all $1 \leq i \leq m$. Therefore the value function $\phi$ is convex. So there is a separating hyperplane at $b$, and so strong duality holds. □

**Remark.** This is useful since we can equivalently solve the dual problem of a linear program to arrive at the same result.

### §3.4.1 The dual of a linear program

Consider the primal problem $P$ in *standard form*:

$$\text{minimise } c^T x \text{ subject to} Ax = b, x \geq 0.$$

The dual problem $D$ is:

$$\text{maximise } g(\lambda) = \inf_{x \in X} L(x, \lambda) \text{subject to } \lambda \in \Lambda.$$

Now we have that

$$\begin{aligned} g(\lambda) &= \inf_{x \geq 0} c^T x - \lambda^T (Ax - b) \\ &= \inf_{x \geq 0} (c^T - \lambda^T A)x + \lambda^T b \end{aligned}$$

Since we need the problem to be bounded, we must have each component of $(c^T - \lambda^T A)$ to be $\geq 0$ (else could just increase the $x_i$). So $\Lambda = \{\lambda : \lambda^T A \leq c^T\}$. So the best choice of $x$ to pick here is the zero vector. Thus $g(\lambda) = \lambda^T b \ \forall \lambda \in \Lambda$. The dual is thus also a linear program:

$$\text{maximise } \lambda^T b \text{ subject to } \lambda^T A \leq c^T.$$

So let's try a linear program in *general form*. We can write it as minimising $c^T x$ subject to $Ax - s = b$, $s \geq 0$ (slack variable). Then

$$\begin{aligned} g(\lambda) &= \inf_{x \in X, s \geq 0} c^T x - \lambda^T (Ax - s - b) \\ &= \inf_{x \in X, s \geq 0} (c^T - \lambda^T A)x + \lambda^T s + \lambda^T b \end{aligned}$$

To ensure our problem is bounded, we require $\lambda^T \geq 0$ (if any component were $\leq 0$, we could simply increase that component of $b$ and $s$). Similarly to before, we must now have $(c^T - \lambda^T A) = 0$ since we can now possibly vary $x$ over all of $\mathbb{R}^n$. Therefore the dual problem is

$$\text{maximise } \lambda^T b \text{ subject to } \lambda^T A = c^T, \lambda \geq 0.$$

This is a linear program in standard form, and so the *dual of the dual is the primal*.

**Proposition 3.4.2**

The primal problem

$$\text{minimise } c^T x \text{ subject to } \begin{cases} a_i^T x \geq b_i & i \in M_1 \\ a_i^T x \leq b_i & i \in M_2 \\ a_i^T x = b_i & i \in M_3 \\ x_j \geq 0 & j \in N_1 \\ x_j \leq 0 & j \in N_2 \\ x_j \text{ free} & j \in N_3 \end{cases}$$

has a dual problem of

$$\text{minimise } p^T b \text{ subject to } \begin{cases} p_i \geq 0 & i \in M_1 \\ p_i \leq 0 & i \in M_2 \\ p_i \text{ free} & i \in M_3 \\ p^T A_j \leq c_j & j \in N_1 \\ p^T A_j \geq c_j & j \in N_2 \\ p^T A_j = c_j & j \in N_3 \end{cases}$$

**Proof.** The proof of this is a problem on Example Sheet 2. □

### §3.4.2 Optimality conditions

If $x$ is feasible for the primal, if $\phi$ is feasible for the dual, and if complementary slackness holds, then $x$ is optimal for the primal and $p$ is the dual-optimal.

**Theorem 3.4.3** (Fundamental theorem of linear programming)

Let $x$ and $p$ be feasible solutions to the primal and dual problem respectively. Then $x$ and $p$ are optimal if and only if:

- $p_i(a_i^T x - b_i) = 0$ for all $i$,
- $x_j(c_j - p^T A_j) = 0$ for all $j$.

(i.e complementary slackness).

**Proof.** Define $u_i = p_i(a_i^T x - b_i)$ and $v_j = x_j(c_j - p^T A_j)$.
Observe that if $x$ and $p$ are primal (dual) feasible, then $u_i \geq 0$ for all $i$ and $v_j \geq 0$ for all $j$. Now

$$\sum_i u_i = \sum_i p_i(a_i^T x - b_i) = p^T Ax - p^T b.$$

Similarly,

$$\sum_j v_j = \sum_i (c_j - p^T A_j)x_j = c^T x - p^T Ax.$$

Therefore $\sum u_i + \sum v_j = c^T x - p^T b$. Hence

$$0 \leq \sum u_i + \sum v_j \leq c^T x - p^T b.$$

**Complementary slackness $\implies$ optimality:**
If complementary slackness holds, then $u_i, v_j = 0$ for all $i, j$. Therefore $c^T x = p^T b$. By weak duality, $x$ and $p$ must be primal and dual optimal, respectively.
**Optimality $\implies$ complementary slackness**:
If $x$ and $p$ are optimal, then strong duality gives us $c^T x = p^T b$. Thus $\sum u_i + \sum v_j = 0$, and since each $u_i, v_j$ is $\geq 0$, we have $u_i, v_j = 0$ for all $i, j$. So complementary slackness holds. □

## §3.5 The simplex method

Recall the solutions to our linear programs are basic feasible solutions, with

$$x = (0, \dots, x_{B_{(1)}}, \dots) \text{ and } x_B = B^{-1}b.$$

The optimality conditions tell us that we could find some kind of algorithm to find a solution, since we could check all of those conditions.

Recall that

$$\text{minimise } c^T x \text{ subject to } Ax = b, x \geq 0 \underset{\text{dual}}{\rightarrow} \text{maximise } \lambda^T b \text{ subject to } \lambda^T A \leq c^T.$$

The optimality conditions are

1. Primal feasible: $Ax = b, x \geq 0$
2. Dual feasible: $A^T \leq c$
3. Complementary slackness: $x^T(c - A^T\lambda) = 0$

Suppose $x$ is a basic feasible solution given by $x_B = (x_{B_{(1)}}, \dots, x_{B_{(m)}})$. Substituting into 3, we get $x_B^T(c_B - B^T\lambda) = 0$. Recall that for a BFS $x$, $x_B > 0$. This means

$$c_B - B^T\lambda = 0 \implies \lambda = (B^T)^{-1}c_B.$$

If this $\lambda$ satisfies $A^T\lambda \leq c^T$, i.e

$$A^T(B^T)^{-1}c_B \leq c^T,$$

then all three optimality conditions are met. The vector $c - A^T(B^T)^{-1}c_B$ is denoted by $\bar{c}$ and is called the vector of **reduced costs**. $\bar{c} \geq 0$ means $x$ is optimal.

**Definition** (Feasible direction)

Let $x \in P$, where $P$ is the feasible set. A vector $d \in \mathbb{R}^n$ is called a **feasible direction** if there exists $\theta > 0$ such that $x + \theta d \in P$.

Let $x$ be a BFS, let $B_{(1)}, \dots, B_{(m)}$ be the indices of the basic variables. Let $B = [A_{B_{(1)}}, \dots, A_{B_{(m)}}]$. $x_B$ is as before. Suppose we move in a direction $d$ such that $d_j = 1$ where $j \notin \{B_{(1),\dots,B_{(m)}}\}$ and $d_i = 0$ for all other nonbasic $i$ where $i \neq j$. It doesn't matter what $\{d_{B_{(1)},\dots,d_{B_{(m)}}}\}$ are. The vector $d$ is the $j$th basic direction:

$$A(x + \theta d) = b \implies Ad = 0 \implies Bd_B + A_j = 0 \implies d_B = -B^{-1}A_j.$$

The $j$th basic direction is thus a feasible direction. How does the cost change when $x \to x + \theta d$? The new cost is

$$c^T(x + \theta d) = c^T x + \theta c^T d = c^T x + \theta(c_j - c_B^T B^{-1} A_j) = c^T x + \theta \bar{c_j}.$$

**Theorem 3.5.1**

Let $x$ be a BFS associated with a basis matrix $B$, and let $\bar{c}$ be the vector of reduced costs. Then $x$ is optimal if and only if $\bar{c} \geq 0$.

**Proof.** Follows from optimality conditions from previous lectures. □

Suppose $x$ is a BFS. If $\bar{c} \geq 0$, then we are done. If $c_j < 0$ for some $j$, then moving in the $j$th feasible direction will reduce the cost. Let $\theta^*$ be the largest possible $\theta$ such that $x + \theta d \geq 0$. We have two cases:

1. If $d \geq 0$, then $\theta^* = \infty$, and the optimal cost is $-\infty$.
2. If $d_i < 0$ for some $i$, we need $x_i + \theta^* d_i \geq 0 \implies \theta^* \leq -\frac{x_i}{d_i}$. We write

$$\theta^* = \min_{i:d_i<0}(-\frac{x_i}{d_i}) = \min_{i\in\{1,2,\ldots,m\}:d_{B_{(i)}}<0}(-\frac{x_i}{d_i}).$$

(For all nonbasic $i$, $d_i$ =0 or 1) Let $l$ be the index minimisng the above term. Let $y = x + \theta^* d$. We know $y$ is feasible with $c^T y < c^T x$.

**Theorem 3.5.2**

$y$ is a basic feasible solution with basis matrix

$$\overline{B} = \begin{pmatrix} \vdots & & \vdots & \vdots & & \vdots \\ A_{B_{(1)}} & \ldots & A_{B_{(l-1)}} & A_j & \ldots & A_{B_{(m)}} \\ \vdots & & \vdots & \vdots & & \vdots \end{pmatrix}.$$

**Proof.** $y$ has exactly $m$ nonzero entries. It's easy to check that $y = x + \theta d$. □

**Definition** (Simplex method)

This is an algorithm to find a solution to a linear program.

1. Start with a BFS $x$ with $B = [A_{B(1)}, \ldots, A_{B(m)}]$.
2. Calculate $\bar{c}$. If $\bar{c} \geq 0$, terminate the algorithm as $x$ is optimal. Otherwise, choose $j$ such that $c_j < 0$.
3. Compute $u = -B^{-1}A_j$. If $u \leq 0$, then cost is $-\infty$, and the algorithm terminates.
4. If some component of $u$ is $> 0$, then set

$$\theta^* = \min_{i=1,2,\ldots,m} \frac{x_{B(i)}}{u_i}.$$

5. Let $l$ be such that $\frac{x_{B(l)}}{e_l} = \theta^*$, set $y = x - \theta^* u$.

### §3.5.1 The full tableau implementation

We define a **simplex tableau**:

| cost, $-c_B^T x_B$ | reduced costs, $\bar{c}$ |
|---|---|
| current BFS, $B^{-1}b$ | reduced costs, $B^{-1}A$ |

.

We follow this algorithm in order to apply the simplex method using the simplex tableau:

1. Start with BFS $x$ and basis matrix $B$.
2. Examine $\overline{c}$ in the "zeroth" row. If $\overline{c} \geq 0$, then we are done. Otherwise choose $j$ such that $\overline{c_j} < 0$.
3. Let $u = B^{-1}A_j$. If $u \leq 0$, the optimal cost is $-\infty$ and we stop the algorithm.
4. Compute $\frac{x_{B(i)}}{u_i}$ for all $i$ such that $u_i > 0$. Let $l$ be such that $\frac{x_{B(l)}}{u_l} = \min_{u_i>0} \frac{x_{B(i)}}{u_i}$.
5. Add to each row of the tableau a constant multiple of the $l$th row so that $u_l$ becomes 1 and all other entries of the pivot column are 0.

# §4 Applications of linear programming

## §4.1 Game theory

We will consider a two-person game. This is defined as two players, P1 and P2. P1 has $m$ possible actions $\{1, 2, \ldots, m\}$. P2 has $n$ possible actions $\{1, 2, \ldots, n\}$. This game works as follows:

If P1 plays $i$ and P2 plays $j$, then P1 wins £ $a_{ij}$ (the entries of a matrix) and P2 loses the same amount. This game is called a "zero-sum game". What is the optimal strategy?

**Definition** (Payoff matrix)

The matrix

$$\begin{pmatrix} a_{11} & \ldots a_{1n} & \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}.$$

Suppose P1 plays first. If P1 picks $i$, then he expects to earn

$$\min_{j \in \{1,\ldots,m\}} a_{ij}.$$

P1 will thus try to solve the problem

$$\text{maximise} \min_{j \in \{1,\ldots,m\}} a_{ij} \text{ subject to } i \in \{1, 2, \ldots, n\}.$$

Now suppose P2 plays first. If P2 picks $j$, he expects P1 to earn

$$\max_{i \in \{1,\ldots,n\}} a_{ij}.$$

P2 will thus try to solve the problem

$$\text{minimise} \max_{i \in \{1,\ldots,n\}} a_{ij} \text{ subject to } j \in \{1, 2, \ldots, m\}.$$

**Example**

If $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$:

- If P1 plays first, then P1 will pick 2, and P2 will pick 1.

- If P2 plays first, then P2 will pick 1, and P1 will pick 2.

When this happens, (2,1) is called a **saddle point** and $a_{21} = 3$ is called the **value** of the game.

Instead, consider the case

$$A = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}.$$

- If P1 plays first, then P1 will pick 1, and P2 will pick 2.
- If P2 plays first, then P2 will pick 2, and P1 will pick 2.

So here we don't have a saddle point.

If we don't have a saddle point, then there's no clear best strategy if the players play simultaneously. This leads us to consider a strategy which uses *randomness*.

### §4.1.1 Mixed strategies

Allow players to choose randomly; P1 picks from a distribution $(p_1, p_2, \dots, p_m)$ and P2 picks from $(q_1, q_2, \dots, q_n)$. If P1 picks from $(p_1, p_2, \dots, p_m)$, the expected reward of P1 if P2 plays $j$ is $\sum_{i=1}^{m} a_{ij} p_i$. So for P1 the optimisation problem we need to solve is

$$\text{maximise } \min_{j \in \{1,\dots,n\}} \sum_{i=1}^{m} a_{ij} p_i \text{ subject to } \sum_{i=1}^{m} p_i = 1 \text{ and } p_i \geq 0 \text{ for } 1 \leq i \leq m.$$

This can be equivalently expressed as follows: if $e = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$, then our problem is

$$\text{maximise } V \text{subject to } A^T p \geq ve, e^T p = 1, p \geq 0.$$

This is a linear program. P2's corresponding linear program is

$$\text{minimise } w \text{ subject to } Aq \leq we, e^T q = 1, q \geq 0.$$

We will show that this is the dual of P1's problem:

Add a slack variable $s \geq 0$ such that $Aq \leq we$ becomes $Aq + s = we$, $s \geq 0$. Then the Lagrangian is

$$L(w, q, s, \lambda_1, \lambda_2) = w(1 - \lambda_1^T e) + (\lambda_1^T A - \lambda_2 e^T)q + \lambda_1 s + \lambda_2..$$

Our feasible set is

$$\Lambda = \left\{ \lambda : \ \lambda_1^T e = 1, \lambda_1^T A - \lambda_2 e^T \geq 0, \lambda_1 \geq 0 \right\}.$$

When $\lambda \in \Lambda$, $\inf L(w, q, s, \lambda_1, \lambda_2) = \lambda_2$, so the dual is exactly the same as P1's problem.

**Theorem 4.1.1**

A strategy $p = (p_1, \dots, p_m)$ is optimal for P1 iff there exist $q$ and $v$ such that

1. $A^T p \geq ve, e^T p = 1, p \geq 0$ (primal feasibility)

2. $Aq \leq ve, e^T q = 1, q \geq 0$ (dual feasibility)

**Proof.** $(p, v)$ and $(q, w)$ are optimal if $(Aq - we)^T p = 0$ and $q^t(A^T p - ve) = 0$, i.e

$$v = w = p^T Aq.$$

□

### §4.1.2 Finding optimal strategies

1. Search for saddle points. If a saddle point exists, then we have solved the problem.
2. Look for dominating actions. Suppose there exists $i$ and $i'$ s.t.

$$a_{ij} \geq a_{i'j} \ \forall j \in \{1, 2, \ldots, n\}.$$

   Then drop row $i'$ from $A$. Do the corresponding removal in the case for $j$ and $j'$.
3. Solve the linear program using the simplex method or any other method.

**Example**

Take the matrix

$$A = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 1 & \frac{1}{2} \\ 1 & 3 & 2 \end{pmatrix}.$$

Then we can note the first row dominates the last, and as such

$$\tilde{A} = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 1 & \frac{1}{2} \end{pmatrix}.$$

P1's strategy is $(p, 1-p, 0)$. The optimisation problem is

maximise $v$ subject to $A^T p \geq v, 2p + 3(1-p) \geq v, 3p + (1-p) \geq v, 4p + \frac{1-p}{2} \geq v$..

The three constraints can be written as

1. $v \leq 3 - p$
2. $v \leq 1 + 2p$
3. $v \leq \frac{1}{2} + \frac{7}{2}p$.

Since this is a one-dimensional problem let's draw a picture:

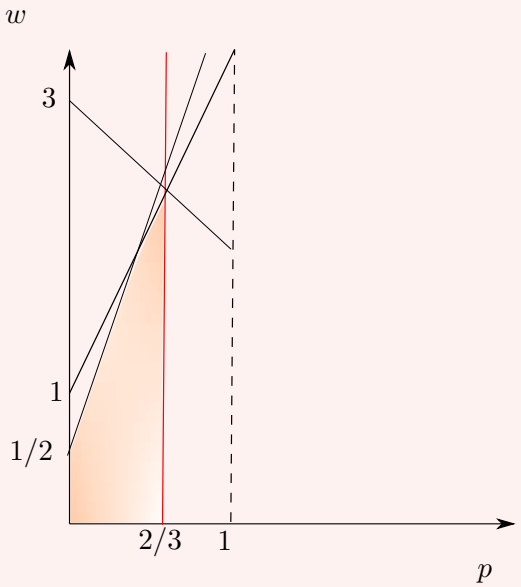



Figure 7: The maximal value is reached inside the shaded area when $p = \frac{2}{3}, w = \frac{7}{3}$.

Thus P1 plays $(\frac{2}{3}, \frac{1}{3}, 0)$ and so $v = \frac{7}{3}$. To solve P2's strategy, we can use complementary slackness. Since $p_1, p_2 \geq 0$, the first two constraints in the dual must be tight. Therefore $Aq = w$. Plugging in,

$$2q + 3(1-q) = \frac{7}{3} \implies q = \frac{2}{3}.$$

So P2's strategy is $(\frac{2}{3}, \frac{1}{3}, 0)$.

## §4.2 Network flows

**Definition** (Graph)

A **directed graph** is defined as a pair of sets of vertices and edges $G = (V, E)$, where $E \subset V \times V$. When $E$ is symmetric ( $(i,j) \in E \leftrightarrow (j,i) \in E$ ) then the graph is said to be undirected. We define $(i,j)$ to be the edge connecting vertex $i$ to vertex $j$.

### §4.2.1 Minimum cost flow on graphs

Given $G = (V, E)$ on $n$ vertices ( $|V| = n$ ), associate to every $(i,j) \in E$ the number $x_{ij}$. $x = \{x_{ij}\}_{(i,j)\in E}$ is called a **flow** on G. The flow $x$ is affected by

1. A vector $b \in \mathbb{R}^n$, where $b_i$ is the amount of flow entering vertex $i$. If $b_i > 0$, we call $i$ a *source*, and if $b_i < 0$, we call $i$ a *sink*.
2. A matrix $C \in \mathbb{R}^{m \times n}$ which gives the cost $c_{ij}$ per unit of flow on $(i,j) \in E$. For a flow of $x_{ij}$ along $(i,j)$ , the cost is $c_{ij}x_{ij}$.
3. Matrices $\overline{M}, M$ determine the lower and upper bounds on $x_{ij}$ for $(i,j) \in E$:

$$M_{ij} \leq x_{ij} \leq \overline{M}_{ij} \ \forall (i,j) \in E..$$

**Definition** (Minimum cost flow)

The minimum cost flow is the linear program

$$\text{minimise} \sum_{(i,j)\in E} c_{ij}x_{ij}$$

subject to:

$$M_{ij} \le x_{ij} \le \overline{M}_{ij} \quad \forall (i,j) \in E$$

$$b_i + \sum_{j:\ (j,i)\in E} x_{ji} = \sum_{j:\ (i,j)\in E} x_{ij} \quad \forall i \in V.$$

The second condition is a conservation law: the flow leaving a vertex must be equal to the flow entering it.

**Definition** (Incidence matrix)

We define a matrix $A \in \mathbb{R}^{|V|\times|E|}$ with rows $v_1, \dots, v_n$ and columns $e_{ij}$, where $1 \le i, j \le n$. $e_{ij}$ represents the node leaving $i$ going to $j$: so $e_{ij}$ has a 1 in position $i$, -1 in position $j$, and 0 elsewhere.

**Remark.** For the linear problem to be feasible, $\sum b_i = 0$.

### §4.2.2 The transport problem

Let's say we have $n$ suppliers and $m$ consumers. The suppliers have capacity $\{s_1, s_2, \dots, s_m\}$ and the consumers have demands $\{d_1, d_2, \dots, d_m\}$. Assume $\sum_{i=1}^n s_i = \sum_{j=1}^n d_j$. The cost of transporting one unit of goods from supplier $i$ to consumer $j$ is $c_{ij}$.

In this example, our graph $G$ is a *bipartite graph*: [pic, all consumers on one side, all suppliers on the other] The optimisation problem we need to solve is

$$\text{minimise} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \text{ subject to } \sum_{j=1}^{m} x_{ij} = s_i \text{ for } i = 1, 2, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = d_i \text{ for } j = 1, 2, \dots, m$$

**Theorem 4.2.1**

Every minimum cost flow problem with finite capacities or non-negative capacities can be recast as an equivalent transport problem.

**Proof.** Consider a min cost flow problem on $G = (V, E)$. We may assume WLOG that $M_{ij} = 0$ for all $(i, j) \in E$. This can be seen by writing $x_{ij} = M_{ij} + \widetilde{x}_{ij}$. Then the conservation equation becomes

$$\widetilde{b}_i + \sum_{(j,i)\in E} \widetilde{x}_{ji} = \sum_{(i,j)\in E} \widetilde{x}_{ij},$$

where $\widetilde{b}_i = \sum_{(j,i)\in E} M_{ji} - \sum_{(i,j)\in E} M_{ij}$. Let's assume going ahead that $M_{ij} = 0$ for all $(i,j) \in E$. If all the costs are $\geq 0$, and if a particular edge has infinite capacity, then we can replace that capacity value by a large number such as $\sum |b_i|$ and not change the optimal solution.

Let's start with some minimum cost flow problem. Our goal is to construct the corresponding transport problem. For every vertex $i$, we create a consumer with demand $\sum_{(i,k)\in E} \overline{m}_{ik} i b_i$. For every edge $(i,j)$, we create a supplier with supply $\overline{m}_i j$. Note that $\sum b_i = 0$ so the total amount of flow in is equal to the total flow out. [picture] The total flow into $i$ is

$$\sum_{k:\ (i,k)\in E} (\overline{m}_{ik} - x_{ik}) + \sum_{k:\ (i,k)\in E} x_{ki} = \sum \overline{m}_{ik} - b_i.$$

Cancelling, this gives the conservation equation for the minimum cost flow problem. We can also easily check that $- \leq x_{ij} \leq \overline{m}_{ij}$. The cost of the flow in the transport problem is $\sum x_{ij} c_{ij}$, which is identical to the cost for a flow $x$ on the minimum cost problem. □

**Theorem 4.2.2**

If for some feasible $x$ we have dual variables $\lambda \in \mathbb{R}^n$ (for suppliers) and $\mu \in \mathbb{R}^m$ (for consumers) such that

$$\lambda_i + \mu_j \leq c_{ij} \text{ and } (c_{ij} - (\lambda_i + \mu_j))x_{ij} = 0 \quad \forall i, j,$$

then $x$ is optimal.

**Proof.**

$$\begin{aligned} L(x,\lambda,\mu) &= \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} - \sum_{i=1}^{n}\lambda_i(\sum_{j=1}^{n} x_j - s_i) - \sum_{j=1}^{m}\mu_j(\sum_{i=1}^{n} x_{ij} - d_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m}(c_{ij} - \lambda_i - \mu_j)x_{ij} + \sum \lambda_i s_i + \sum \mu_j d_j. \end{aligned}$$

□

### §4.2.3 The transport algorithm

The optimality conditions for $x$ are that there exists $\lambda_1, \ldots, \lambda_n$ (Lagrange variables for suppliers) and $\mu_1, \ldots, \mu_n$ (Lagrange variables for consumers) satisfying

$$\begin{aligned} \lambda_i + \mu_j &\leq c_{ij} \quad \forall i,j \text{ (dual feasibility)} \\ (c_{ij} - \lambda_i - \mu_j)x_{ij} &= 0 \quad \forall i,j \text{ complementary slackness .} \end{aligned}$$

Note that we can shift $\lambda_1, \mu_1 \mapsto \lambda_1 - c, \mu_1 + c$ for any constant $c$ so we can pick $\lambda_1 = 0$ WLOG.

To solve the transport problem we are going to have a 'transportation tableau':

| | | | |
|---|---|---|---|
| $\lambda_1+\mu_1, x_{11}, c_{11}$ | $\lambda_1+\mu_2, x_{12}, c_{12}$ | $\dots$ | $\lambda_1+\mu_m, x_{1m}, c_{1m}$ |
| $\lambda_2+\mu_1, x_{21}, c_{21}$ | $\lambda_2+\mu_2, x_{22}, c_{22}$ | $\dots$ | $\lambda_2+\mu_m, x_{2m}, c_{2m}$ |
| $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $\lambda_n+\mu_1, x_{n1}, c_{n1}$ | $\lambda_n+\mu_2, x_{n2}, c_{n2}$ | $\dots$ | $\lambda_n+\mu_m, x_{nm}, c_{nm}$ |

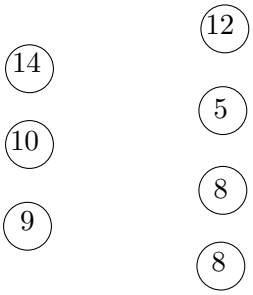
How do we construct the initial transportation tableau?

1. Take the first supplier and try and transport as much as possible to the first consumer (increase $x_{11}$ until either $s_1$ or $d_1$ is satisfied)
2. If the supplier is satisfied, we go to the next supplier and repeat step 1.
3. If the demand is satisfied, move to the next consumer. Repeat step 1.

This process gives our initial 'naive' basic feasible solution.

In order to see how to use this, let's look at an example. Suppose our initial graph is

**Suppliers** **Consumers**




The cost matrix for this problem is given to be

$$C = \begin{pmatrix} 5 & 3 & 4 & 6 \\ 2 & 7 & 4 & 1 \\ 5 & 6 & 2 & 4 \end{pmatrix}.$$

We know that since $x_{ij} > 0$ for all $i, j$ we have $\lambda_i + \mu_j = C_{ij}$. Solving these five equations gives us

$$\lambda_1 = 0, \mu_1 = 5, \mu_2 = 3, \lambda_2 = 4, \mu_3 = 0, \lambda_3 = 2, \mu_4 = 2.$$

**Theorem 4.2.3**

Let $x$ be a basic feasible solution formed by the earlier process. Then the set of edges with strictly positive flow form a connected graph that has no cycle. This graph $T$ is a spanning tree with exactly $m + n - 1$ edges.

**Proof.** The proof is beyond the scope of this course; we simply state this theorem, so that we know that this always happens! □

Now let's do the transport tableau as in the example.

| | | | |
|---|---|---|---|
| $5, 12, 5$ | $3, 2, 3$ | $0, 0, 4$ | $2, 0, 6$ |
| $9, 0, 2$ | $7, 3, 7$ | $4, 7, 4$ | $6, 0, 1$ |
| $7, 0, 5$ | $5, 0, 6$ | $2, 1, 2$ | $4, 8, 4$ |

If $c_{ij} \geq \lambda_i + \mu_j$ for all $i, j$ then $x$ is optimal. If $(i,j) \notin T$ such that $c_{ij} < \lambda_i + \mu_j$ then $(i,j)$ and the edges of $T$ of form a loop. Then we increase $x_{ij}$ until $x_{ab} = 0$ for some $(a,b) \in T$. Then we update $\lambda, \mu$ and repeat until we get an optimal value.

### §4.2.4 Max flow - min cut

Consider a network consisting of $n$ nodes labelled $1, \ldots, n$ and directed edges between them with capacities $c_{ij}$ on the arc from node $i$ to node $j$. Let $x_{ij}$ denote the flow in the arc $i \to j$, where $0 \leq x_{ij} \leq c_{ij}$.

Our problem is

maximise $\delta($ the flow $)$

$$\begin{aligned}
\text{subject to} \quad & \sum_{\{j:\ (i,j)\in E\}} x_{ij} - \sum_{\{j:\ (i,j)\in E\}} x_{ji} = 0 \quad \forall i \in \{2,3,\ldots,n-1\} \\
& \sum_{\{j:\ (i,j)\in E\}} x_{1j} - \sum_{\{j:\ (i,j)\in E\}} x_{j1} = \delta \quad \forall i \in \{2,3,\ldots,n-1\} \\
& \sum_{\{j:\ (i,j)\in E\}} x_{nj} - \sum_{\{j:\ (i,j)\in E\}} x_{jn} = -\delta \quad \forall i \in \{2,3,\ldots,n-1\} \\
& 0 \leq x_{ij} \leq c_{ij} \quad \forall (i,j) \in E.
\end{aligned}$$

In this problem, we don't have any costs associated with the flow; the problem is essentially finding the maximum flow through the graph.

**Definition** (Cut)

A cut of a graph $G$ is a partition of its vertices into two sets $S$ and $V \backslash S$. The capacity of a cut is given by

$$\mathcal{C}(S) = \sum_{\{(i,j)\in E:\ i\in S, j\in V\backslash S\}} c_{ij}.$$

**Theorem 4.2.4**

Let $x$ be a feasible flow to the max flow problem with value $\delta$. Then for any cut $(S, V\backslash S)$ such that $1 \in S$, $n \in V\backslash S$ we have $\delta \leq \mathcal{C}(S)$.

**Proof.** For any two sets $X, Y \subseteq V$ define

$$f_x(X,Y) = \sum_{\{(i,j)\in E:\ i\in X, j\in Y\}} x_{ij}.$$

Let $(S, V\backslash S)$ be a cut s.t $1 \in S$, $n \in V\backslash S$. Then summing the flow conservation conditions in $S$, we have

$$\delta = \sum_{i\in S} \left( \sum_{\{j:\ (i,j)\in E\}} x_{ij} - \sum_{\{j:\ (i,j)\in E\}} x_{ji} \right)$$

$$
\begin{aligned}
&= f_x(S, V) - f_x(V, S) \\
&= f_x(S, S) + f_x(S, V\backslash S) - f_x(S, S) - f_x(V\backslash S, S) \\
&= f_x(S, V\backslash S) - f_x(V\backslash S, S) \\
&\leq f_x(S, V\backslash S) && \leq \mathcal{C}(S).
\end{aligned}
$$

□

**Theorem 4.2.5** (Max flow - min cut theorem)

Let $\delta^*$ be the value of the max. flow. Then we have

$$\delta^* = \min\{\mathcal{C}(S) : \ 1 \in S, n \in V\backslash S, S \subseteq V\}.$$

**Proof.** A path $v_0, v_1, \ldots, v_k$ is a sequence of vertices such that for every pair of adjacent vertices is connected by an edge in either direction.

A path is called an **augmenting path** if every 'forward' edge satisfies $x_{ij} < c_{ij}$ and every 'reverse' edge satisfies $x_{ij} > 0$. Augmenting paths are special because we can tweak them to increase the flow. An optimal flow cannot have augmenting paths from 1 to $n$.

If $x$ is optimal, we define

$$S = \{1\} \cup \{i : \ \exists \text{ an augmenting path from 1 to } i\}.$$

Note that $n \in V\backslash S$, so this is a valid cut. We have

$$\delta^* = f_x(S, V\backslash S) - f_x(V\backslash S, S) = f_x(S, V\backslash S) = \mathcal{C}(S).$$

□

The algorithm we used is called the Ford-Fulkerson algorithm:

1. Start with a feasible flow (say $x = 0$)
2. Find an augmenting path; if no such path exists, $x$ is optimal.
3. Increase the flow along the path as much as possible.
4. Repeat back to step 1.

**Example**

Take a graph with six vertices: [picture] [do later when i can be bothered]

**Proposition 4.2.6**

If all capacities are rational, then the Ford-Fulkerson algorithm always finds the best flow in a finite number of steps.

**Proof.** First scale the whole problem WLOG so all capacities are integers. Then the flow increases by at least 1 at each step. □

### §4.2.5 The bipartite matching problem

We take a $k$-regular bipartite graph of size $2n$, i.e a bipartite graph where there are $n$ nodes on each side and each edge has $k$ connections. Our problem is to match the left and right nodes bijectively using the edges given.

**Theorem 4.2.7**

Every $k$-regular bipartite graph has a perfect matching.

**Proof.** First construct two new vertices on the left and right that are connected to their respective sides of the bipartite graph, and initialise a max-flow problem as in the image: [picture] For every edge, we attach a flow of $\frac{1}{k}$. This is clearly a feasible flow. Note that we know by considering the cut $\{1\}$ that the max flow for this new graph is $\frac{n}{2}$. □

**Remark.** End of notes!