Contents

1 Convex optimization

1.1 Terminology

As the name suggests, our goal is to minimize or maximize a function $f(x)$ subject to some constraints. However, we look at minimizing because to maximize we just minimize the negative.

We consider if $f(x) \in \mathbb{R}$ with $x \in \mathbb{R}^n$ with h a function from $\mathbb{R}^n$ to $\mathbb{R}^m$, and h is subject to constraints like $y_i \leq b_i$. We call f the objective function. A constraint on h is called a functional constraint and we can have a regional constraint like $f(x) \in A \subseteq \mathbb{R}^n$.

The x^* such that $f(x^*)$ is the minimum value is called the optimal solution of course.

The set of allowed x's subject to constraints is called the feasible set.

Definition. A slack variable s is an extra variable we can introduce to turn an inequality $h(x) \leq b$ to $h(x) + s = b$ where $s \geq 0$. We will see why this is useful later.

1.2 Convex functions

Definition. A set $S \subseteq \mathbb{R}^n$ is said to be convex if any 2 points in S have the property that the line connecting them is contained in S. Specifically, for x, y in S, and all $\lambda \in [0, 1]$, $\lambda x + (1 - \lambda) y \in S$.

A set is said to be concave if it is not convex. See figure 1 for a demonstration.

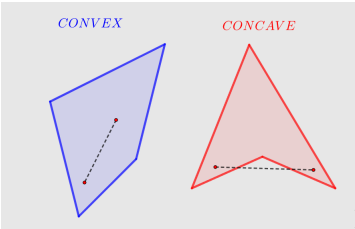

Figure 1

Definition. A function $f : S \to \mathbb{R}$ is said to be convex if for every $x, y \in S$ and $\lambda \in [0, 1]$,

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

Also, f is said to be concave if $-f$ is convex. $\log(x)$ is an example of a concave function. Note that the domain has to be convex for a function to be convex.

Basically the line has to be above the curve, see figure 2.

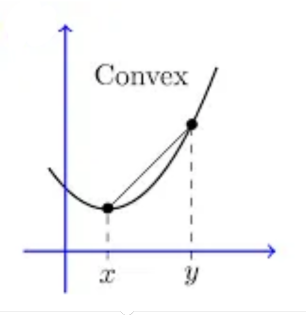

Figure 2

Example. x^2 and $|x|$ are convex functions.

Note that every function of the form $ax + b$ is both convex and concave.

Proposition. If f is a differentiable function from $\mathbb{R} \to \mathbb{R}$ then it is convex if its derivative is increasing

Proof. Obvious from looking at it.

□

Theorem. If f is a differentiable function from $\mathbb{R}^n \to \mathbb{R}$ then $f(y) \geq f(x) + (y - x) . \nabla f(x)$ for all x, y in $\mathbb{R}^n$ if and only if f is convex.

Note that if we have a point where $\nabla f(x) = 0$ this theorem would immediately imply that x is a minimiser of f.

Proof. If f is convex and $t \in [0, 1]$ then

$$f(x + t(y - x)) \leq (1-t) f(x) + t f(y)$$

This means that

$$f(y) \geq \frac{f(x + t(y-x)) - (1-t) f(x)}{t} = f(x) + \frac{f(x + t(y-x)) - f(x)}{t}$$

By properties of the grad, we note that in the limit as $t \to 0$, the $\frac{f(x+t(y-x))-f(x)}{t}$ term approaches $(y - x) . \nabla f(x)$. Taking the limit on both sides gives the inequality we want.

Conversely, suppose f satisfies this inequality. Set $X_t = (1-t) x + ty$. Then by the inequality we are assuming, we have

$$f(x) \geq f(X_t) + \nabla f(X_t) . (x - X_t)$$

$$f(y) \geq f(X_t) + \nabla f(X_t) . (y - X_t)$$

We can multiply the first inequality by $1-t$ and the second by t and add them together. This gives

$$(1-t) f(x) + t f(y) \geq f(X_t) + (1-t) \nabla f(X_t) . (x - X_t) + t \nabla f(X_t) . (y - X_t)$$

Note that

$$(1-t)(x - X_t) = t(1-t)(x-y) = -t(y - X_t)$$

so the grad terms cancel. Therefore we have

$$(1-t) f(x) + t f(y) \geq f(X_t)$$

This is exactly the convexity inequality.

□

If f is a twice differentiable function $\mathbb{R} \to \mathbb{R}$, then note that it is convex if and only if $f''(x) \geq 0$ everywhere.

Definition. In the context of optimization we denote the hessian matrix of f as $\triangle f(x)$ where $\triangle f(x)_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$

Let f be 2 times differentiable. Recall from differential equations that

$$f(y) = f(x) + (y - x) . \nabla f(x) + \frac{1}{2} (y-x)^T \triangle f(x) (y - x) + o\left((y-x)^2\right)$$

Theorem. A twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if $\triangle f(x) \geq 0$ for all $x \in \mathbb{R}^n$.

Proof. For any x, y we have, by Taylor's theorem, that for some z on the line from x to y, the following equality is exact.

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

If the hessian is always positive then the first order condition from above holds which implies that f is convex. This is because the hessian is positive definite and we can diagonalize it and all the eigenvalues will be non-negative and the result follows.

As for the converse, we do this in lecture 2 of variational principles.

□

[Lecture 1 ends]

1.3 Gradient descent algorithm

Observe that $f(y) \approx f(x) + (y - x) . \nabla f(x)$ for y close to x.

Now start at x_0 and set $t = 0$.

Now find the grad direction (to find the direction in which the function value is decreasing) given $\varepsilon > 0$ (ie, a step size), we move by this step size in the descent direction. Ie, we move by $-\varepsilon \nabla f$ and repeat.

Eventually, we decide that if the gradient is sufficiently small we will stop.

This could go wrong: If our function is very sharp (in the sense of very large second derivative), and our step size is too large, we could overshoot the minimum point. The obvious way to avoid this is to take a small step size. However, if our step size is too small the algorithm might take too long.

1.4 Smoothness and strong convexity

Definition. A twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be β-smooth and α-strongly convex if $\alpha I \leq \triangle f \leq \beta I$. Here we compare matrices by saying $A \leq B$ if $B - A$ has no negative eigenvalues. These conditions rule out the situations above where the algorithm might fail.

Theorem. Let f be β-smooth, then

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

If f is α- strongly convex then

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

Essentially we're saying the function is trapped between 2 quadratics that pass through the same point and have the same tangent at that point.

Proof. Using the multivariate Taylor series expansion, we have (by Taylor's theorem), for some z on the line between x and y,

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

Now because $\beta I \geq \triangle f$, $\beta I - \triangle f$ is positive (semi)definite so we will certainly have the following inequality, which completes the proof:

$$(y - x)^T \alpha I (y - x) \leq (y - x)^T \triangle f(z) (y - x) \leq (y - x)^T \beta I (y - x)$$

□

Corollary. Let f be β-smooth, then $f\left(x - \frac{1}{\beta} \nabla f(x)\right) \leq f(x) - \frac{1}{2\beta} |\nabla f|^2$

Proof. Substitute in the last theorem $y = x - \frac{1}{\beta} \nabla f(x)$

□

This says that if we have a step size of $\frac{1}{\beta}$ you will not overshoot since you will actually get smaller on each step.

Corollary. Let f be α-strongly convex and let x^* be the minimizer of f, then $f(x^*) \geq f(x) - \frac{1}{2\alpha} |\nabla f(x)|^2$

Proof. By the theorem,

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

Now clearly,

$$f(x^*) \geq f(x) + \nabla f(x) \cdot (x^* - x) + \frac{\alpha}{2} |x^* - x|^2 \geq \min_y \left[f(x) + \nabla f(x) \cdot (y - x) + \frac{\alpha}{2} |y - x|^2 \right]$$

Now the grad of the RHS wrt y is $\nabla f(x) + \alpha (y - x)$, so we set $y^* := x - \frac{1}{a} \nabla f(x)$, and if we plug in this y^* into the above equation in place of "minimum over y" then the result follows.

□

Remark. We know that if $|\nabla f|^2 < 2\alpha\varepsilon$ then we are within ε of the minimum.

This means that for sufficiently nice functions we can pick a step size that gurantees we get smaller on each step and get fairly close to the minimum.

Theorem. Let $\beta > \alpha > 0$ and let f be α-strongly convex and β-smooth. Then the gradient descent with step size $\frac{1}{\beta}$ satisfies at iteration T

$$f(x_T) - f(x^*) \leq e^{-\frac{T\alpha}{\beta}} (f(x_0) - f(x^*))$$

which means that in fact the convergence is exponential. For some reason this is generally called linear convergence.

Proof. Recall that $x_{t+1} = x_t - \frac{1}{\beta}\nabla f(x_t)$. Then we know from a previous corrolary that

$$f(x_{t+1}) \leq f(x_t) - \frac{1}{2\beta} |\nabla f(x_t)|^2$$

Therefore

$$f(x_{t+1}) - f(x^*) \leq f(x_t) - f(x^*) - \frac{1}{2\beta} |\nabla f(x_t)|^2$$

However, by another previous corrolary, $f(x_t) - f(x^*) \leq \frac{1}{2\alpha} |\nabla f(x_t)|^2$ which means that

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

Therefore, we have

$$f(x_{t+1}) - f(x^*) \leq \left(1 - \frac{\alpha}{\beta}\right) (f(x_t) - f(x^*)) \leq e^{-\frac{\alpha}{\beta}} (f(x_t) - f(x^*))$$

Since $(1 - x) \leq e^{-x}$ for all x.

Now, the error multiplies by at most $e^{-\frac{\alpha}{\beta}}$ on each step so by step T it will be multiplied by at most $e^{-\frac{T\alpha}{\beta}}$.

□

Example. Let $f(x, y) = \frac{1}{2}\left(x^2 + 100y^2\right)$, then we have to use $\alpha = 1$ and $\beta = 100$ so we need a step size of 0.01. But here $\frac{\alpha}{\beta}$ is small so we will converge slowly. However, in this case if we rescale y, then this will improve it a lot.

[Lecture 2 ends]

1.5 Newton's method

The idea of Newton's method is to look at the second order Taylor expansion instead of the first. The second order taylor expansion is

$$f(y) = f(x) + \nabla f(x) . (y - x) + \frac{1}{2} (y - x)^T \triangle f(x) (y - x) + o\left((y - x)^2\right)$$

We will do a method where x_{t+1} is the value where the minimum value of the above Taylor expansion (truncated after the quadratic term) about x_t is attained. We are minimizing the local quadratic approximation.

To minimize $f(x) + \nabla f(x) . (y - x) + \frac{1}{2} (y - x)^T \triangle f(x) (y - x)$ over y, we will find the gradient with respect to y. The first term is a constant, and we get

$$\nabla f(x) + \triangle f(x) (y - x)$$

Therefore $y = x - \triangle f(x)^{-1} \nabla f(x)$ is the minimizer. So the rule for Newton's method is

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

Note that strong convexity implies the hessian isactually invertible.

Note that this is similar to the Newton Raphson method in the sense that finding the root of something is identical to trying to find a local minimum of the antiderivative using the above method. If f is a 1-variable function then we set $f' = g$ then the grad is the first derivative and the hessian is the second derivative, so we get the rule

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

to find a root of g, exactly the formula for the Newton Raphson procedure.

Definition. Recall that we can define the norm of a matrix A as the length of the largest vector it can send a unit vector to.

Note: The following (for the proof of the error bound) was not lectured other than the statement of the bound, which is

$$f(x_k) - f(x^*) \leq \frac{2\alpha^3}{l^2}\left(\frac{l}{2\alpha^2}|\nabla f(x_0)|\right)^{2^{k+1}}$$

Lemma. Let f be twice continuously differentiable and α-strongly convex, then we have the following bound on the operator norm of its inverse hessian:

$$\left\|(\triangle f(x))^{-1}\right\| \leq \frac{1}{\alpha}$$

Proof. The hessian is symmetric so its eigenvalues are real and it is diagonalizable. By strong convexness, $\triangle f(x) - \alpha I$ has only non-negative Eigenvalues, and is positive semidefinite, ie for any non-zero vector v,

$$v^T \triangle f(x) v \geq \alpha v^{Tv} = \alpha |v|^2$$

Also, by considering the orthonormal basis of the eigenvector directions, we see that $\triangle f$ is diagonal and none of its entries can be less than α without contradicting strong-convex-ness so the eigenvalues of $\triangle f$ are at least α. Therefore, the eigenvalues of $(\triangle f)^{-1}$ are at most $\frac{1}{\alpha}$.

Now we are considering what a diagonal matrix with each entry positive and at most $\frac{1}{\alpha}$ can send a unit vector to, we see that it will not send a unit vector to anything of length more than $\frac{1}{\alpha}$, so that is an upper bound for the operator norm.

□

Lemma. Let f be a twice continuously differentiable function from $\mathbb{R}^n \to \mathbb{R}$, then the following holds:

$$\nabla f(x_{k+1}) = \nabla f(x_k) + \int_0^1 \triangle f(x_k + t(x_{k+1} - x_k))(x_{k+1} - x_k)\, dt$$

Proof. Define $\gamma(t) = x_k + t(x_{k+1} - x_k)$. Now define $G(t) = \nabla f(\gamma(t))$. Note that component-wise,

$$G_i(t) = \frac{\partial f}{\partial x_i}(x_k + t(x_{k+1} - x_k))$$

Now by the multivariate chain rule,

$$\frac{d}{dt}G_i(t) = \sum_{j=1}^{n} \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}(\gamma(t))\frac{d\gamma_j(t)}{dt}\right)$$

Therefore,

$$\frac{d}{dt}G_i(t) = \sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(\gamma(t))(x_{k+1} - x_k)_j$$

Therefore, by matrix multiplication,

$$\frac{d}{dt}G(t) = \triangle f(\gamma(t))(x_{k+1} - x_k)$$

Therefore by the fundamental theorem of calculus the integral of this is $G(1) - G(0)$ so the result follows.

□

Lemma.

$$|\nabla f(x_{k+1})| \leq \frac{l}{2\alpha^2}|\nabla f(x_k)|^2$$

Proof. We will prove that $|\nabla f(x_{k+1})| \leq \frac{l}{2\alpha^2} |x_{k+1} - x_k|^2$

We know f is also twice continuously differentiable by the Lipschitz hessian condition. Recall that the rule is

$$x_{k+1} = x_k - \triangle f(x_k)^{-1} \nabla f(x_k)$$

By continuity of the second derivative, terms involving the hessian are integrable, so we can safely apply the previous lemma. We know that

$$\nabla f(x_{k+1}) = \nabla f(x_k) + \int_0^1 \triangle f(x_k + t(x_{k+1} - x_k))(x_{k+1} - x_k)\, dt$$

But by our newton step, we can write

$$\nabla f(x_k) = -\triangle f(x_k)(x_{k+1} - x_k) = -\int_0^1 \triangle f(x_k)(x_{k+1} - x_k)\, dt$$

Therefore, substituting this into the previous equation,

$$\nabla f(x_{k+1}) = \int_0^1 [\triangle f(x_k + t(x_{k+1} - x_k)) - \triangle f(x_k)](x_{k+1} - x_k)\, dt$$

Taking the absolute value on both sides and using the magic inequality about the operator norm (which I will denote with double bars),

$$|\nabla f(x_{k+1})| \leq \int_0^1 \|\triangle f(x_k + t(x_{k+1} - x_k)) - \triangle f(x_k)\| \, |x_{k+1} - x_k|\, dt$$

Because the hessian is l-Lipschitz, we have

$$\|\triangle f(x_k + t(x_{k+1} - x_k)) - \triangle f(x_k)\| \leq lt\, |x_{k+1} - x_k|$$

Therefore we get

$$|\nabla f(x_{k+1})| \leq \int_0^1 lt\, |x_{k+1} - x_k|\, |x_{k+1} - x_k|\, dt = \frac{l}{2} |x_{k+1} - x_k|^2$$

Now also we have by operator norm properties,

$$|x_{k+1} - x_k| \leq \left\| (\triangle f(x_k))^{-1} \right\| |\nabla f(x_k)| \leq \frac{1}{\alpha} |\nabla f(x_k)|$$

The last inequality is by a previous lemma. Substituting this into our inequality for $|\nabla f(x_{k+1})|$ we get

$$|\nabla f(x_{k+1})| \leq \frac{l}{2\alpha^2} |\nabla f(x_k)|^2$$

□

Theorem. If f is a twice differentiable α-strongly convex function that has an l-lipschitz Hessian, i.e. for all x and y, $|\triangle f(x) - \triangle f(y)| \leq l|x - y|$ with respect to the operator norm, then Newton's method satisfies

$$f(x_k) - f(x^*) \leq \frac{2\alpha^3}{l^2} \left(\frac{l}{2\alpha^2} |\nabla f(x_0)| \right)^{2^{k+1}}$$

Note that this is very good because it is double-exponential, but the catch is we need to be at a good enough point at the beginning.

Proof. Take the inequality from the previous lemma and multiply both sides by $\frac{l}{2a^2}$ and define $\phi_k := \frac{l}{2\alpha^2} |\nabla f(x_k)|$. Then we have proved that

$$\phi_k \leq \phi_0^{2^k}$$

It therefore follows that

$$|\nabla f(x_k)| \leq \frac{2a^2}{l} \left(\frac{l}{2a^2} |\nabla f(x_0)| \right)^{2^k}$$

By strong convexity, we have that

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

Minimizing the right hand side with respect to y gives that $\nabla f(x) + \alpha(y-x) = 0$ which implies that

$$y - x = -\frac{1}{\alpha}\nabla f(x)$$

Therefore,

$$f(y) \geq f(x) - \frac{1}{\alpha}|\nabla f(x)|^2 + \frac{\alpha}{2}\left|-\frac{1}{\alpha}\nabla f(x)\right|^2 = f(x) - \frac{1}{2\alpha}|\nabla f(x)|^2$$

In particular,

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

Substituting in our expression for $|\nabla f(x_k)|$ and setting $x = x_k$ we get

$$f(x_k) - f(x^*) \leq \frac{1}{2\alpha}\left[\frac{2a^2}{l}\left(\frac{l}{2a^2}|\nabla f(x_0)|\right)^{2^k}\right]^2$$

$$f(x_k) - f(x^*) \leq \frac{2a^3}{l^2}\left[\left(\frac{l}{2a^2}|\nabla f(x_0)|\right)^{2^k}\right]^2 = \frac{2a^3}{l^2}\left(\frac{l}{2a^2}|\nabla f(x_0)|\right)^{2^{k+1}}$$

□

This is useful in the context of machine learning (or it was like 10 years ago but things are complicated now). However, those applications use functions of millions of variables, where the hessian would therefore have trillions of variables. This is not practical.

1.6 Barrier method

Suppose we want to minimize subject to constraints like $a.x \leq b$ so you are restricted to within that set.

The issue is that previously our metods might take us outside the box.

The trick we can do is to minimise $f(x) + \sum_{i=1}^{m}\phi(a.x-b)$ where $\phi(x) = -\log(-x)$, which essentially will make you step away from the boundary. However, the minimum of this function is not the minimum that we want.

To do this, we introduce a parameter $t > 0$ and the problem of minimizing over $x \in \mathbb{R}^n$ the function $tf(x) + \sum_{i=1}^{m}\phi(a.x-b)$. If $t$is big we are giving more weight to f. We keep solving the problem with increasing t and we hopefully get closer to the minimum. The steps are as follows:

1. Find an x in the domain not on the boundary and set $t > 0$, $a > 1$

2. Compute the minimum of $tf(x) + \sum_{i=1}^{m}\phi(a.x-b)$ using Newton's method with x as a starting point.

3. Repeat after multiplying our t by α and stop when t is large enough or so large that we fall out of the domain.

[Lecture 3 ends]

2 Lagrange multiplier method

2.1 The method

Consider minimizing f(x)subject to $x \in X$ and $h(x) = b$ where $h : \mathbb{R}^n \to \mathbb{R}^m$.

The idea of this is to transform a constrained problem into an unconstrained problem. In theory we could take $X = \{h(x) = b\}$ but that would defeat the purpose.

Define the Lagrangian as follows:

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

Now if $h(x) > b$ we essentially get a penalty, so instead of minimizing f subject to constraints we will try to minimize L.

Note: If one vector is $>$ another vector then in this course we mean all components are greater.

Theorem. (Lagrange sufficiency) Let $x^* \in X$ where and satisfying the constraint so that $h(x^*) = b$ and there is $\lambda^* \in \mathbb{R}^m$ such that $L(x^*, \lambda^*) = \min_{x \in X} L(x, \lambda^*)$, then x^* minimizes $f(x)$ subject to $h(x^*) = b$.

Proof. Clearly $\min_{x \in X, h(x)=b} f(x) \leq f(x^*)$ since x^* is in the feasible set.

Also,

$$\min_{x \in X, h(x)=b} f(x) = \min_{x \in X, h(x)=b} (f(x) - \lambda^*.(h(x) - b))$$

since $\lambda.(h(x) - b) = 0$ in the feasible set. Now this is

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

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

Therefore $\min_{x \in X, h(x)=b} f(x) = f(x^*)$ if we can find such a λ^*.

□

Example. We want to minimize $-x_1 - x_2 + x_3$ subject to $x_1^2 + x_2^2 = 4$, $x_1 + x_2 + x_3 = 1$. This is the intersection of a cylinder and a plane, which is some ellipse.

Here,

$$h(x) = \begin{pmatrix} x_1^2 + x_2^2 \\ x_1 + x_2 + x_3 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

Now the Lagrangian

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

$$= (1 + \lambda_2)x_1 - \lambda_1 x_1^2 - (1 + \lambda_2) x_2 - \lambda_1 x_2^2 + (1 - \lambda_2) x_3 + 4\lambda_1 + \lambda_2$$

By the previous theorem, if we can find any λ and then successfully minimize $L(x)$ in all of $x = \mathbb{R}^n$, we will be done. Note that we need $\lambda_2 = 1$ otherwise by making x_3 sufficiently small we would never minimize the Lagrangian. Similarly λ_1 has to be negative otherwise we would have a curving down quadratic in x_1 which has no minimum attained.

For the pair (λ_1, λ_2) subject to these constraints we can simaltaneously minimize the terms in x_1, x_2 and x_3 using basic calculus. We get that it does not matter what x_3 is and that $x_1 = x_2 = \frac{-1}{\lambda_1}$. Now we need to pick λ such that this minimizer actually satisfies the constraints in order to satisfy the conditions of the theorem. Since $x_1 = x_2$ and $x_1^2 + x_2^2 = 4$, we get that $x_1 = x_2 = \sqrt{2}$.

By the constraints we can find that $\lambda_1 = \frac{-1}{\sqrt{2}}$ and that $x_3 = 1 - 2\sqrt{2}$.

The λ used above is called a Lagrange multiplier.

We will now solve problems subject to $h(x) \leq b$. To do this, we will add a slack variable. We will say $h(x) + s = b$ and $s \geq 0$ are our constraints.

The Lagrangian is

$$L(x, \lambda, s) = f(x) - \lambda.(h(x) + s - b)$$

We will define a new set:

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

For each $\lambda \in \Lambda$ we will find an x^* and s^* (if they exist) such that

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

We then just need to find any λ^* such that x^* and s^* satisfy the constraints.

In the process above we want to minimize the lagrangian subject to $s \geq 0$. Suppose that for a particular value of λ we do this and we arrive at a solution x^* and s^*. Then if

$$\lambda = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{pmatrix}$$

then if any $\lambda_i > 0$ this means that we can pick some large s in that component to make L as small as we want and so $\lambda \notin \Lambda$. Therefore, to minimize the Lagrangian, if any $\lambda_i = 0$, we would want to make $s_i = 0$ to minimize the increase in the value of f. Therefore, we know that the right choice of s that minimizes the Lagrangian will satisfy $\lambda_i s_i = 0$ for each i. Therefore, we have 2 cases:

We are minimizing $f(x) - \lambda.(h(x) - b)$ now. Either $h(x)_i = b_i$, in which case any $\lambda_i \leq 0$ is fine, or $h(x)_i > b_i$ in which case to minimize, we need $\lambda_i = 0$, since otherwise this violates the constraint that $\lambda_i s_i = 0$ for each i (not summed over), since s is defined to be $h(x)_i - b_i$.

Example. We will minimize $x_1 - 3x_2$ subject to $x_1^2 + x_2^2 \leq 4$ and $x_1 + x_2 \leq 2$. This is minimizing a linear function along a slightly cut off circle (See figure 3), where one could find the minimum by considering the most extreme lines of constant $x_1 - 3x_2$ that intersect the region - We will see this intuition in linear programming. However, we will do this using the theory above.

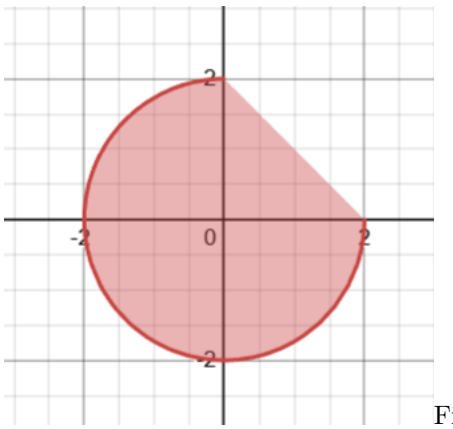

Figure 3

We will transform the problem into

minimize $x_1 - 3x_2$

subject to $\begin{cases} x_1^2 + x_2^2 + s_1 = 4 & s_1 \geq 0 \\ x_1 + x_2 + s_2 = 2 & s_2 \geq 0 \end{cases}$

We will now take the Lagrangian and try to find optimal x and s.

Note that right now, we are not subject to constraints. We are minimizing L subject only to $s \geq 0$ and at the end we will worry about satisfying the constraints. We have

$$L = (x_1 - 3x_2) - \lambda_1 \left(x_1^2 + x_2^2 + s_1 - 4\right) - \lambda_2 (x_1 + x_2 + s_2 - 2)$$

$$= \left((1-\lambda_2)\,x_1 - \lambda_1 x_1^2\right) - \left((-3-\lambda_2)\,x_2 - \lambda_1 x_2^2\right) - \lambda_1 s_1 - \lambda_2 s_2 + (4\lambda_1 + 2\lambda_2)$$

As discussed, we must have $\lambda_1, \lambda_2 \leq 0$ and $\lambda_1 s_1 = \lambda_2 s_2 = 0$ at the optimum. We can minimize each term independently for x_1 and x_2 by differentiating.

For x_1 and x_2 respectively, we have

$$1 - \lambda_2 - 2\lambda_1 x_1 = 0 \tag{1}$$

$$-3 - \lambda_2 - 2\lambda_1 x_2 = 0 \tag{2}$$

If $\lambda_1 = 0$ these cannot both be true, ie we would have that L is unbounded below! Since x ranges over all of $\mathbb{R}^2$.

Therefore we have $\lambda_1 < 0$ and therefore $s_1 = 0$. We note that if $\lambda_2 < 0$ then $s_2 = 0$. We note that if we have picked the "right" lagrange multiplier then we will have that our x_1 and x_2 satisfy

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

$$x_1 + x_2 = 2$$

since $s_1 = s_2 = 0$ so we don't have any +s in the constraints.

This implies that

$$(x_1, x_2) = (0, 2) \text{ or } (2, 0)$$

If it is $(0, 2)$ then solving equations (1) and (2) will give $\lambda_1 = 1$ which is impossible. If it is $(2, 0)$ then we get $\lambda_2 = 1$ which is also impossible. Therefore, the only possibility is that $\lambda_2 = 0, \lambda_1 \leq 0$.

This is complementary slackness, in the sense that it is always the case that if one variable is slack (defined by an inequality), the other will be tight (defined by an equality).

Now since we know that $s_2 = 0$ leads to a contradiction but that we must have $s_1 = 0$, and $\lambda_2 = 0$, we have

$$1 - 2\lambda_1 x_1 = 0$$

$$-3 - 2\lambda_1 x_2 = 0$$

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

$$x_1 + x_2 + s_2 = 2$$

The first 2 equations give $x_1 = \frac{1}{2\lambda_1}$, $x_2 = \frac{-3}{2\lambda_1}$. Substituting into the third equation, we can solve a quadratic for λ_1 and we get (by the fact that it is negative) that we must have $\lambda_1 = -\sqrt{\frac{5}{8}}$. Therefore, we can find x_1 and x_2 which are optimal by Lagrange sufficiency. We get

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

Intuition: The reason that $s_1 = 0$ is because the optimal solution is on the circle, and the reason $s_2 \neq 0$ is because the optimal solution is not on the line segment.

I am, of course, going to include a picture that shows intuitively why this method works, I really don't know why the lecturer doesn't show this because it is really cool.

Suppose we want to minimize $x^2 + y^2$ subject to $y = 1$. Then this is a paraboloid, see figure 4.

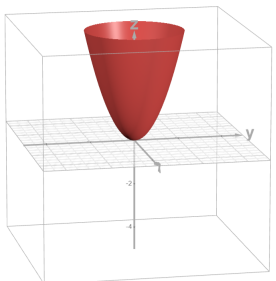

Figure 4

We see that the global minimum is attained when $y = 0$, which is not what we want. However, if we look at $x^2+y^2-2(y-1)$ (Graph in figure 5), then we notice that

1. The behavior when $y = 1$ does not change

2. The global minimum is now at $y = 1$

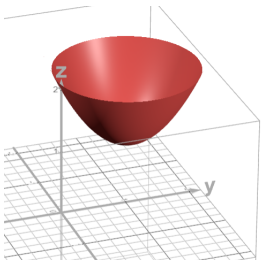

Figure 5

This is the picture you should have in mind for why Lagrange multipliers work. You can even imagine continuously deforming λ and watching the graph change until it ends up with the global minimum at a feasible point, where the value at the feasible points never changes.

[Lecture 4 ends]

We will now try to determine when this method works.

Note that in the above method we have $n + 2m$ variables (x, λ, s), we set some m of them to 0, and we have $n + m$ equations (n minimizations, m constraints), so the number of equations matches the number of variables. This makes sense, since that is usually what we want heuristically.

2.2 Duality

Consider first the problem of minimizing $f(x)$ subject to $h(x) = b$. Here, we define the dual objective function $g : \Lambda \to \mathbb{R}$ as follows:

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

We observe that often this dual problem is easier to solve than the original problem. For every fixed x, the Lagrangian L is linear in λ for each fixed x. Therefore, the function above is taking the minimum of a bunch of planes. We see (by considering a picture) that the minimum of a bunch of planes is a concave function, provided the following lemma holds.

Lemma. The domain of g is actually convex.

Proof. We want to show that the set where $\inf_{x \in X} L(x, \lambda) > -\infty$ is convex. Given $\lambda_1, \lambda_2 \in \Lambda$ it suffices to show that for all $\alpha \in [0, 1]$, $\alpha\lambda_1 + (1-\alpha)\lambda_2 \in \Lambda$.

Now to do this,

$$L(x, \alpha\lambda_1 + (1-\alpha)\lambda_2) f(x) - (\alpha\lambda_1 + (1-\alpha)\lambda_2).(h(x) - b)$$
$$= \alpha(f(x) - \lambda_1.(h(x) - b)) + (1-\alpha)(f(x) - \lambda_2.(h(x) - b)) = \alpha L(x, \lambda_1) + (1-\alpha) L(x, \lambda_2)$$

By definition,

$$g(\alpha\lambda_1 + (1-\alpha)\lambda_2) = \inf_{x \in X} [\alpha L(x, \lambda_1) + (1-\alpha) L(x, \lambda_2)] \geq \alpha g(\lambda_1) + (1-\alpha) g(\lambda_2) > -\infty$$

So the result follows.

□

The dual problem is to maximise $g(\lambda)$ subject to $\lambda \in \Lambda$. The primal problem is the original problem we want to solve.

Theorem. (Weak duality theorem) If x is in the feasible set and $\lambda \in \Lambda$ then $f(x) \geq g(\lambda)$. This therefore implies that

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

Note that if we find a pair (x, λ) such that the above inequality is an equality then this implies that x solves the original problem.

Proof. For any $\lambda \in \Lambda$ and $x \in X(b)$, we have that

$$f(x) \geq \inf_{x \in X(b)} f(x) = \inf_{x \in X(b)} f(x) - \lambda.(h(x) - b) \geq \inf_{x \in X} L(x, \lambda) = g(\lambda)$$

Therefore the result follows.

□

Definition. The difference $\inf_{x \in X(b)} f(x) - \sup_{\lambda \in \Lambda} g(\lambda)$ is called the duality gap.

Definition. We say a problem has strong duality if this duality gap is 0.

Definition. I'm going to make up this definition to do this rigorously. We say a problem has "Very strong duality" if it has strong duality and the values $\inf_{x \in X(b)} f(x)$ and $\sup_{\lambda \in \Lambda} g(\lambda)$ are both attained.

Theorem. The Lagrange method works if and only if a problem has very strong duality.

Proof. If the Lagrange method works this means that there exists a pair (x^*, λ^*) such that $h(x^*) = b$ and

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

We say min and not inf because one of the assumptions of saying the Lagrange method works is that the minimum is actually attained.

This therefore implies that $g(\lambda^*) = f(x^*)$, so we have strong duality.

Conversely, if we have strong duality, then the Lagrange method will work by using the λ where equality is attained.

□

Definition. A function $\phi : \mathbb{R}^m \to \mathbb{R}$ is said to have a supporting hyperplane at a point $b \in \mathbb{R}^m$ if there exists a λ such that for all $c \in \mathbb{R}^m$ we have that

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

Intuitively, this says that $\phi(c)$ always lives above some m-dimensional plane through $(b, \phi(b))$.

Definition. Define the value function $\phi : \mathbb{R}^m \to \mathbb{R}$ as

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

where

$$X(c) = \{x : h(x) = c, x \in X\}$$

Theorem. The problem of minimizing f subject to $h(x) = b$ has very strong duality if the values $\inf_{x \in X(b)} f(x)$ and $\sup_{\lambda \in \Lambda} g(\lambda)$ are both attained for each possible b and ϕ has a supporting hyperplane at b.

Conversely, if strong duality holds for a suitable λ, then λ gives the supporting hyperplane.

This theorem is easy to understand from the "picture" of the Lagrange method I showed in these notes.

Proof. The proof is deferred to later in this lecture

□

Note that clearly supporting hyperplanes always exist for convex functions by taking $\lambda = \nabla f(b)$.

Theorem. The following problem has a convex value function ϕ: Minimize $f(x)$ subject to $h(x)_j \leq b_j$ where X (the domain of f) is a convex set, f is convex, and the h_j are also convex functions.

Proof. Let b_1 and b_2 be 2 points in the domain of ϕ, meaning the corresponding feasible sets are non-empty. Let $\alpha \in [0,1]$.

By definition of infimum, for any εwe can find points x_1, x_2 such that $h(x_1) \leq b_1$ and $f(x_1) \leq \phi(b_1) + \varepsilon$, and similarly for x_2,b_2.

Now define $x_\alpha = \alpha x_1 + (1-\alpha) x_2$. Since the domain of X is given to be a convex set this is allowed.

Now since the constraint function h is convex, we know that

$$h(x_\alpha) \leq \alpha h(x_1) + (1-\alpha) h(x_2) \leq \alpha b_1 + (1-\alpha) b_2$$

Thus, x_α is feasible for the optimization problem defining $\phi(\alpha b_1 + (1-\alpha) b_2)$

Also, at any x_α, since ϕ is the infimum, we have

$$\phi(\alpha b_1 + (1-\alpha) b_2) \leq f(x_\alpha)$$

By convexity of f, and the definition of x_1 and x_2 from earlier,

$$\phi(\alpha b_1 + (1-\alpha) b_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2) \leq \alpha\phi(b_1) + (1-\alpha)\phi(b_2) + \varepsilon$$

Since this holds for every positive ε the result follows.

□

We will now prove equivalence between a supporting hyperplane and strong duality.

Proof. Suppose the value function has a supporting hyperplane at b and the relevant inf and sup is attained.

Then we have

$$g(\lambda) = \min_{x \in X} f(x) - \lambda.(h(x) - b) = \min_{C} \min_{x \in X(C)} f(x) - \lambda.(h(x) - c) - \lambda.(c - b)$$
$$= \min_{C} \phi(c) - \min_{x \in X(C)} \lambda.(h(x) - c) + \lambda.(c - b)$$

If $x \in X(C)$ then the term $\lambda.(h(x) - c)$ is 0 so we have

$$g(\lambda) = \min_{C} \phi(c) - \lambda.(c - b) \geq \phi(b)$$

by assumption.

However, by weak duality, $g(\lambda) \leq \phi(b)$ so they are equal so we have very strong duality at b.

Conversely suppose strong duality holds for suitable λ, then by definition of ϕ,

$$\phi(b) = g(\lambda) = \inf_{x \in X} f(x) - \lambda.(h(x) - b) = \inf_{x \in X} f(x) - \lambda.(h(x) - c) - \lambda.(c - b) \leq \phi(c) - \lambda.(c - b)$$

□

This also has a cool application to economics. Suppose a factory owner produces n types of products from m types of materials, and the vector of how much they produce is $(x_1, x_2, ..., x_n)$ and the profit is a sufficiently nice function $f(x)$. We then define $h_j(x)$ to be the amount of raw material j consumed when making products x. If the factory owner has b_i of these raw materials, then they will maximize $f(x)$ subject to $h \leq b$.

Now suppose a supplier offers $\varepsilon = (\varepsilon_1, \varepsilon_2, ..., \varepsilon_m)$ additional raw materials, then the amount of additional profit will be equal to

$$\phi(b+\varepsilon) - \phi(b)$$

If we decide to not worry about technical details for a second, then we will see that

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

Therefore, what happens is that this derivative is the price of material j. We call these "shadow prices", since they are not visible to the outside world and depend on the internal state of the factory. We will call the vector of prices $\lambda = \nabla \phi$.

Now what happens is if a particular raw material was not used up there is a slack variable so the shadow price is 0, and if we have used it all up then the slack variable is 0, so this is the complementary slackness we saw earlier.

The dual problem is trying to maximize profit for the seller who charges a price for their raw materials and buys the finished product from the factory. Their profit will be the cost of materials minus what they get from buying products, which is

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

What is interesting is that the seller is trying to maximize this and the factory owner is trying to minimize it.

[Lecture 5 ends]

3 Linear programming

3.1 Basic properties

Definition. A linear program is when f and h are both linear and we want to minimize f subject to $h \leq b$**.**

More generally, we want to minimize $c.x$ subject to $a_i.x$ greater then or equal to, or less than or equal to, or equal to b_i. We can also have constraints like $x_j \geq 0$ or $x_j \leq 0$. Specifically, we will have conditions like

$$a_i.x \geq b_i, i \in M_1$$

$$a_i.x \leq b_i, i \in M_2$$

$$a_i.x = b_i, i \in M_3$$

$$x_j \geq 0, j \in N_1$$

$$x_j \leq 0, j \in N_2$$

$$x_j \text{ free}, j \in N_3$$

We can actually reduce this to a matrix A, such that we are minimizing $c.x$ subject to $Ax \geq b$. For M_1 do nothing, for M_2 make that row $-a_i$, for M_3 make an a_i and $-a_i$ row, for N_1 put 1 in the j'th position of the row and 0 everywhere else, for N_2 do that but -1 instead of 1, for N_3 put no row. You get the point.

A linear program written as minimize $c.x$ subject to $Ax = b, x \geq 0$ is said to be in standard form.

If we instead have $Ax \leq b$ with no constraint on x this is called canonical form.

We can get a problem into standard form by changing x into $x^+ - x^-$ and requiring they both be positive (we can get any vector by subtracting 2 positive vectors) and adding a slack variable s that we also say must be positive. We can concatenate (x^+, x^-, s) into a 3* larger vector and make A 3* longer so it reads $(A, -A, I)$ and then we will have a problem in standard form.

Proposition. The dual of the problem is maximizing $b.\lambda$ subject to $\lambda \in \Lambda$

Proof. Adding slack variables, we have

$$L(x, s, \lambda) = c.x - \sum_{i \in M_1} \lambda_i (A_i.x - s_i - b_i) - \sum_{i \in M_2} \lambda_i (A_i.x - s_i - b_i) - \sum_{i \in M_3} \lambda_i (A_i.x - b_i)$$

We don't include the N constraints because we consider any point that violates those constraints to not be in the domain in the first place, in the sense that they are not in what we call X.

We note that N_1, N_2, N_3 partition the dimensions of the domain. We will denote the j'th column of the matrix A to be A_i. We then reorder the terms in the sums above and we get

$$= \sum_{j \in N_1} (c_j - \lambda.A_j) x_j + \sum_{j \in N_2} (c_j - \lambda.A_j) x_j + \sum_{j \in N_3} (c_j - \lambda.A_j) x_j + \sum_{i \in M_1} \lambda_i s_i - \sum_{i \in M_2} \lambda_i s_i + \sum_{i \in M_1 \cup M_2 \cup M_3} \lambda_i b_i$$

In order to ensure that this cannot be made to go to $-\infty$, Λ is the set of λ subject to the following constraints:

$$\Lambda = \begin{cases} \lambda_i \geq 0 & i \in M_1 \\ \lambda_i \leq 0 & i \in M_2 \\ \lambda.A_j \leq c_j & j \in N_1 \\ \lambda.A_j \geq c_j & j \in N_2 \\ \lambda.A_j = c_j & j \in N_3 \end{cases}$$

If $\lambda \in \Lambda$ then this is minimized when all x_j are 0 since we never have any reason to make any of them smaller (eg, the first term is a positive thing times a positive thing which is minimized if $x_j = 0$ and similar for the second and third), and similarly it will be minimized whenever the slack variables are 0. Therefore, the minimum of the Lagrangian is just

$$\sum_{i \in M_1 \cup M_2 \cup M_3} \lambda_i b_i = b.\lambda$$

Now by definition of the dual problem, it is indeed to maximize $b.\lambda$ subject to $\lambda \in \Lambda$.

□

It is possible to show that the dual of the dual problem is the primal problem but thankfully we will not use this so we don't need to prove it.

Theorem. Let x and λ be feasible solutions to the primal problem and its dual. Then these are both optimal if and only if $(c_j - \lambda.A_j) x_j = 0$ for all j and $\lambda_i (A_i.x - b_i) = 0$ for all i.

Note that by this theorem, given some x, if we solve for λ and it is feasible, we know we have the optimal solution.

Proof. Define $u_i = \lambda_i (a_i.x - b_i)$, $v_j = (c_j - \lambda.A_j) x_j$.

By feasibility, we know that these are always non-negative.

Now we get

$$\sum_i u_i = \lambda^T A x - \lambda^T b$$

$$\sum_j v_j = c^T x - \lambda^T A x$$

Therefore,

$$\sum_i u_i + \sum_j v_j = c^T x - \lambda^T b$$

By weak duality we know that since $f(x) = c^T x$ and $g(x) = \lambda^T b$, this is always non-negative. Ie, $c^T x \geq \lambda^T b$.

Now if they are equal we have the optimal solution for the primal and the dual, and since the u_i's and v_j's are non-negative and their sum would be 0, they would each be 0. So this proves the first direction.

Conversely, if $u_i = 0$ for all i and $v_j = 0$ for all j then $c^T x = \lambda^T b$ which implies by weak duality that they are optimal.

□

Definition. Let C be a convex set, then $x \in C$ is called an extreme point if it does not lie on a line between two points that are not equal to eachother.

We think of this as the boundary.

Proposition. The maximum of a convex function always occurs at an extreme point, if it occurs at all.

Proof. At a non-extreme point x, we have that $x = \delta y + (1 - \delta) z$ for some points y and z not equal to x, and therefore by convexity

$$f(x) \leq \delta f(y) + (1 - \delta) f(z) \leq \max(f(y), f(z))$$

However, this is not a complete proof unless we know that we can reach an extreme point by going to the ends of lines. However, we can reach a boundary point and this will be an extrene point unless there is a line on the boundary. To make this into a proper proof,we do this by induction on the dimension. Clearly, it is true in 1 dimension.

For this step, we will use your mental picture of what a convex set is.

Suppose that this works for all dimensions $< k$. Then in k dimensions we reach a boundary point when we draw a line from x, and if there is a line from x where the boundary point we reach is not an extreme point then f at that point is $\geq f(x)$ and consider the set of lines that lie in the convex set that go through said boundary point and not as an endpoint, then this is a convex set with lower dimension than k. It helps to see why by an example: If we are in a cube and we draw a line from an interior point and reach the faces then we would reach a boundary point the lines from which would be a square which is a convex set of lower dimension. Then f restricted to that boundary convex set achieves its maximum at an extreme point of that convex set, which is an extreme point of the main set, and therefore f is larger than $f(x)$ at an extreme point of the main set.

□

Therefore this proves the intuition that to solve a linear program (which can be written as maximizing a convex function) we just need to look at extreme points. In fact, we can just look at the value at the vertices of our region, which is nice.

[Lecture 6 ends]

Now we will talk about actually solving linear programs. a standard form program, ie, minimizing $c.x$ subject to $Ax = b$ and $x \geq 0$, where A has m rows (for constraints) and n columns.

We will make some assumptions for now:

1. Rows of A are linearly independent. This implies $n \geq m$ because row rank equals column rank.

2. Every set of m columns of A is linearly independent.

3. To state assumption 3 we need to do some definitions first.

Assumption 1 can be made without loss of generality because any row dependent on the others has its value dotted with x depending on the value of the others, so b would have to be a specific value for that to even be possible. However, assumption 2 cannot be made without loss of generality.

Definition. A point x that satisfies $Ax = b$ and has at most m non-zero entries is called a basic solution. A solution x that satisfies $x \geq 0$ is called a basic feasible solution.

Intuitively, if we have more than $n - m$ constraints to 0 we will have more than n constraints total for n variables so it will be more constraints than variables.

Now to find any basic solution, pick $B(1), B(2), ..., B(m)$ such that x of all other indices is 0. We now have

$$Ax = \sum_{i=1}^{n} A_i x_i = \sum_{i=1}^{m} A_{B(i)} x_{B(i)}$$

We are therefore solving an equation $Bx = b$ where B is $m^* m$. This B is called the basis matrix, it is A restricted to the $B(i)$ columns of A. By linear independence assumptions, this is solvable.

Note that by picking the B's differently, we can get $\begin{pmatrix} n \\ m \end{pmatrix}$ basic solutions. However, we can eliminate the ones with vector solutions that are not entirely non-negative and we will have the basic feasible solutions. This gives us an algorithm to solve linear programs but it is easy to see that this algorithm is not fast, since $\begin{pmatrix} n \\ m \end{pmatrix}$ can get big.

Now assumption 3 is that every basic feasible solution has *exactly* m non-zero entries.

Theorem. A vector x is a basic feasible solution if and only if it is an extreme point of the feasible set (ie, $Ax = b, x \geq 0$)

Proof. First suppose x is a basic feasible solution that is not an extreme point meaning that it can be written as $x = (1 - \delta)\, y + \delta z$. Then we hope to show that we must have $y = z$. If for some index, $x_i = 0$, then since y, z are non-negative, then $y_i = z_i = 0$.

We now define $y_B = \begin{pmatrix} Y_{B(1)} \\ \vdots \\ Y_{B(m)} \end{pmatrix}$ and z_B similarly. We then have $By_B = b$ and $Bz_B = b$. Therefore, $y_B = z_B = B^{-1} b$, thus $x = y = z$.

Conversely, suppose x is not a basic feasible solution, then x must have more than m non-zero indices. Let these be $i_1, i_2, ..., i_r$ where $r > m$. Then the columns $A_{i_1}, ..., A_{i_r}$ are linearly dependent since the rank of A is m. Therefore, we can find weights that are not all 0 such that we have

$$w_{i_1} A_{i_1} + w_{i_2} A_{i_2} + ... + w_{i_r} A_{i_r} = 0$$

We now define w to be w_{i_j} if $i = i_j$ and 0 for all things not in $i_1, ..., i_r$. We now have a non-zero vector w with $Aw = 0$. Now given x, there is an ε such that $x \pm \varepsilon w$ are both feasible since it can be moved in the w direction without the indices becoming negative, ie x is not extreme.

□

3.2 The simplex method

The idea is to jump from vertex to vertex.

When the linear program is in standard form the optimality conditions are

1. Primal feasibility: $x$is feasible

2. Dual feasibility: Since $x_i \geq 0$ for all i this implies $c_j \geq \lambda . A_j$ for λ to be in Λ. See the previous theorem where we proved the dual was maximizing $b.\lambda$, where now we note that every x is in N_1 by standard form.ff

3. $(c_j - \lambda . A_j)\, x_j = 0$ for all j, by a previous theorem about conditions for optimality. It is sufficient as we already have the "x is feasible" condition. We call this condition complementary slackness, and to be honest I'm not sure why it is the same as the complementary slackness from earlier but whatever.

Condition 3 is only worth checking for $j \in \{B_1, B_2, ..., B_n\}$

This therefore reduces to $(c_B - \lambda.B)\, x_B = 0$ where x_B is x restricted to the B indices and similar for c.

Since $x_B > 0$, we must have $c_B = B^T\lambda$ to satisfy condition 3 so we can solve for λ. Now condition 2 holds as long as $c_j \geq \left(B^T\right)^{-1} c.A_j$. In this case, x is optimal.

We now check this for each basic feasible solution and can stop when we find an optimal one without checking the rest.

[Lecture 7 ends]

Now recalling, the optimality conditions above, we will construct a new algorithm.

Note that the optimality condition is $c_j \geq \left(B^T\right)^{-1} c.A_j$.

Definition. The vector of reduced costs is defined to satisfy $\bar{c}^T = c^T - c_B^T B^{-1} A$. Note that $\bar{c}_j = c_j - c_B^T B^{-1} A_j$. Since this is the transpose of $c_j - \left(B^T\right)^{-1} c.A_j$ which is ≥ 0 exactly for optimal solutions, it therefore follows that $\bar{c} \geq 0$ is our optimality condition.

Now start with $x = \left(X_{B(1)}, X_{B(2)}, ..., X_{B(m)}, 0, ..., 0\right)$. If this is not optimal then for some j^*, $\bar{c_{j^*}} < 0$.

Now pick some non-basic j and decide we want to make $x_j > 0$ while keeping all other non-basic x indices non-zero. Ie, we want to perturb x in a direction

$$\left(d_{B(1)}, d_{B(2)}, ..., d_{B(m)}, 0, ..., 0, 1, 0, ..., 0\right)$$

Since we need to still satisfy $Ax = b$ we need to have $Bd_b + A_j = 0$ so that $A\left(x + t\left(d_B, 0, ..., 0, 1, 0, ..., 0\right)\right) = A\left(x\right)$.

Now we know that $d_B = -B^{-1}A_j$. Now we will look at the cost at the point after we move in this direction.

This would be

$$c.\left(x + te_j - tB^{-1}A_j\right) = c.x + t\left(c_j - c_B^T B^{-1} A_j\right) = c.x + t\bar{c_j}$$

This means that if $\bar{c_j} < 0$ to begin with this will reduce the c.x thing that we are trying to minimize.

We want to make t as large as possible while keeping the solution still feasible, ie we want to make none of the $X_{B(i)} < 0$. The only issue is if some $d_{B(i)} < 0$ - if this is never the case we can make $t = -\infty$ and we stop the algorithm and say that there is no solution. The most we can move is $-\frac{X_{B(i)}}{d_{B(i)}}$, so we will make t the smallest such value over all i from 1 to m where $d_{B(i)} < 0$.

Say this minimum is achieved at l. Then the $B(l)$ index would become 0 and the j index would become non-zero. If there is a tie between 2 $\frac{X_{B(i)}}{d_{B(i)}}$ values then the algorithm gets stuck (However one of our assumptions is that this is never the case), otherwise we end up at a new better basic feasible solution. That is the essence of the simplex algorithm.

Since the cost reduces every time, we will not get trapped in a cycle and will eventually get to the optimal point.

The simplex algorithm starts with what is called a Simplex tableau. We are sitting at a basic feasible point x and this is what it looks like:

$-c_B^T x_B$	$\bar{c_1}\ \bar{c_2}\ \bar{c_3}...\bar{c_n}$
$x_{B(1)}$	$B^{-1}A_1$,$B^{-1}A_2$,...,$B^{-1}A_n$
$x_{B(2)}$	
$\vdots$	
$x_{B(m)}$	

3.3 Applying the simplex algorithm

The algorithm, starting at a standard form problem and basic feasible solution x and basis matrix B is as follows.

Repeat:

Pick j such that $\bar{c_j} < 0$ (If there are none, x is optimal. We often pick the smallest one, but we don't have to.)

Set d to $-B^{-1}A_j$

If $d \geq 0$ optimal cost is $-\infty$ so terminate algorithm

Set θ^* to $\min_{d_i<0} -\frac{x_{B(i)}}{d_i}$

Note: It is confusing but almost always if I say d_i or d_l I mean $d_{B(i)}$ or $d_{B(l)}$, since i and l are reserved for indices in the B world, not the A world.

Set l to the i that gave the minimum in the previous step

Set x to $x + \theta^* d$ and turn j into a basis element instead of $B(l)$ since the set of non-zero indices has changed accordingly.

So that is the algorithm. We can do it by drawing a table like above. We modify the algorithm so that instead of setting x to $x + \theta^* d$ we do row operations (ie, adding constant multiples of rows to themselves or other rows) in the table so that $d_{B(l)}$(ie, the l row of the j column where the j column is d_B by construction) becomes 1 and everything else in that column becomes 0.

After this we replace the l row with a j row since j is now part of the B family.

Theorem. The above modification does not change the outcome of the algorithm

Note that the proof was not lectured, as although it is not conceptually difficult, there are many cases to check and a lot of algebra.

Proof. **Step 1.** Demonstrate it for the first column.

In the first version of the algorithm: For the leaving row $i = l$, $x_{B(l)}$ becomes 0, and for all other rows, it comes $x_{B(i)} + \theta^* d_i$. The entering variable x_j becomes θ^*.

In the modified version: The leaving row l is divided by d_l to become 1. Therefore, the leftmost entry in this row, which is $x_{B(l)}$ becomes $-\frac{x_{B(l)}}{d_{B(l)}} = \theta^*$. However, it is also the case that the new x_j is θ^* so when we replace the index of the row this actually matches.

Now for all $i \neq l$, to eliminate every other row in that column, we note that we need to add $d_{B(i)}$ times the l row to the i row, so in the leftmost column $x_{B(i)}$ will change by $\theta^* d_{B(i)}$. This matches the modified x from the first version of the algorithm.

To get the j entry in the first row to 0 we have to subtract $\bar{c_j}$ times the new row l from row 0. Since the top left entry is the cost (Since only the x_B entries are non-zero) and it changes by $\theta^* \bar{c_j}$ after the row operation, it works because that is (as we showed earlier) exactly the amount that the actual cost changes by when we do the original algorithm.

Step 2. Demonstrate it for the interior of the Tableau.

Any other column, say column k, represents the vector $y_k = B^{-1}A_k$. We want to prove that the row operations transform the column y_k into $B_{\text{new}}^{-1}A_k$. Note that the new basis matrix just has the $B(l)$ column replaced with the A_j column, specifically

$$B_{new} = B + \left(A_j - A_{B(l)}\right) e_l^T$$

It follows that

$$B^{-1}B_{\text{new}} = I + \left(B^{-1}A_j - B^{-1}A_{B(l)}\right) e_l^T$$

Since $A_{B(l)}$ is the l-th column of B, we know $B^{-1}A_{B(l)} = e_l$. We also know that $B^{-1}A_j$ is the j-th column in the current Tableau which we call y_j. We get that $B^{-1}B_{\text{new}} = I + (y_j - e_l) e_l^T$

Define $M := B^{-1}B_{\text{new}}$. Notice that M is mostly the identity matrix except the column in the l position is y_j.

Now, let E represent the matrix that the algorithm left multiplies the table by. Specifically, E satisfies $Ey_j = e_l$ (However, E is not unique). Also, since E only adds multiples of l to other rows, it follows that for all $k \neq l$, $Ee_k = e_k$. Because E maps the columns of M to the identity matrix, it follows that

$$E = M^{-1} = \left(B^{-1}B_{\text{new}}\right)^{-1}$$

Therefore,

$$E\left(B^{-1}A_k\right) = B_{new}^{-1}BB^{-1}A_k = B_{\text{new}}^{-1}A_k$$

This means the interior has the desired result. Now we just need to deal with the top column (reduced costs).

Step 3. Demonstrate it for the top column (reduced costs)

To do this, note that $E = M^{-1} = \left(I + (d - e_l)\, e_l^T\right)^{-1}$ since $B^{-1}A_{B(l)} = e_l$ from earlier and also $B^{-1}A_j = d$ from earlier. I claim that

$$\left(I + (d - e_l)\, e_l^T\right)^{-1} = I - \frac{1}{d_l}(d - e_l)\, e_l^T$$

To prove this, we will compute

$$\left(I + (d - e_l)\, e_l^T\right)\left(I - \frac{1}{d_l}(d - e_l)\, e_l^T\right)$$

$$= I - (d - e_l)\, e_l^T \frac{1}{d_l}(d - e_l)\, e_l^T + (d - e_l)\, e_l^T - \frac{1}{d_l}(d - e_l)\, e_l^T$$

$$= I - \frac{1}{d_l}de_l^T de_l^T + \frac{1}{d_l}de_l^T e_l e_l^T + \frac{1}{d_l}e_l e_l^T de_l^T - \frac{1}{d_l}e_l e_l^T e_l e_l^T + de_l^T - e_l e_l^T - \frac{1}{d_l}de_l^T + \frac{1}{d_l}e_l e_l^T$$

$$= I - \frac{1}{d_l}de_l^T de_l^T + \frac{1}{d_l}de_l^T + \frac{1}{d_l}e_l e_l^T de_l^T - \frac{1}{d_l}e_l e_l^T + de_l^T - e_l e_l^T - \frac{1}{d_l}de_l^T + \frac{1}{d_l}e_l e_l^T$$

$$= I - de_l^T + \frac{1}{d_l}de_l^T + e_l e_l^T - \frac{1}{d_l}e_l e_l^T + de_l^T - e_l e_l^T - \frac{1}{d_l}de_l^T + \frac{1}{d_l}e_l e_l^T$$

$$= I$$

Therefore $E = I - \frac{1}{d_l}(d - e_l)\, e_l^T$. Note that $c_{B_{\text{new}}} = c_B + \left(c_j - c_{B(l)}\right) e_l$.

Now let k be any arbitrary column. Then

$$\bar{c_k}_{\text{new}} = c_k - c_{B_{\text{new}}}^T B_{\text{new}}^{-1}A_k = c_k - \left(c_B^T + \left(c_j - c_{B(l)}\right) e_l^T\right) B^{-1}A_k$$

$$= c_k - \left(c_B^T + \left(c_j - c_{B(l)}\right) e_l^T\right) Ey_k = c_k - c_B^T Ey_k - \left(c_j - c_{B(l)}\right) e_l^T Ey_k$$

Lets evaluate the second term.

$$e_l^T E = e_l^T\left(I - \frac{1}{d_l}(d - e_l)\, e_l^T\right) = e_l^T - \frac{d_l - 1}{d_l}e_l^T = \frac{1}{d_l}e_l^T$$

Therefore

$$e_l^T Ey_k = \frac{1}{d_l}e_l^T y_k$$

We will deal with this in a minute. Lets evaluate the first term.

$$c_B^T Ey_k = c_B^T\left(I - \frac{1}{d_l}(d - e_l)\, e_l^T\right) y_k$$

$$= c_B^T y_k - \frac{1}{d_l}c_B^T (d - e_l)\left(e_l^T y_k\right)$$

$$= c_B^T y_k - \frac{1}{d_l}\left(c_B^T d - c_B^T e_l\right)(y_k)_l$$

Recall that $d = B^{-1}A_j$. Since

$$\bar{c}_j = c_j - c_B^T B^{-1}A_j = c_j - c_B^T d$$

we know that $c_B^T d = c_j - \bar{c_j}$. Also, $c_B^T e_l = c_{B(l)}$. We can substitute these in to get

$$c_B^T E y_k = c_B^T y_k - \frac{1}{d_l}\left(c_j - \bar{c_j} - c_{B(l)}\right)(y_k)_l$$

Now recall that what we were doing is dealing with the fact that

$$\bar{c_k}\text{new} = c_k - c_B^T E y_k - \left(c_j - c_{B(l)}\right) e_l^T E y_k$$

But we have a lot of information about these terms that we can substitute back. Specifically,

$$\bar{c_k}\text{new} = c_k - c_B^T y_k + \frac{1}{d_l}\left(c_j - \bar{c_j} - c_{B(l)}\right)(y_k)_l - \left(c_j - c_{B(l)}\right)\frac{1}{d_l} e_l^T y_k$$

$$= c_k - c_B^T y_k + \frac{1}{d_l}\left(c_j - c_{B(l)}\right)(y_k)_l - \frac{1}{d_l}\bar{c_j}(y_k)_l - \frac{1}{d_l}\left(c_j - c_{B(l)}\right) e_l^T y_k$$

$$= \left(c_k - c_B^T y_k\right) - \bar{c_j}\left[\frac{1}{d_l}(y_k)_l\right]$$

Since the original reduced cost was in fact $\bar{c_k} = c_k - c_B^T y_k$, the change is $-\bar{c_j}\left[\frac{1}{d_l}(y_k)_l\right]$.

Step 4. Show that this change $-\bar{c_j}\left[\frac{1}{d_l}(y_k)_l\right]$ is exactly what the change was in the reduced costs from the algorithm before the modification.

Note that for a B entry,

$$\bar{c}_{B(i)} = c_{B(i)} - c_B^T B^{-1} A_{B(i)} = c_{B(i)} - c_B^T B^{-1} B_i = c_{B(i)} - c_B^T e_i = 0$$

Therefore we have a vector that is given by $\bar{c}^T = c^T - \lambda^T A$ for some λ and is also 0 for all basis entries. Since we have that $\bar{c}_B = 0$ we need λ to satisfy $\lambda^T B = c_B^T$ by restricting the above equation to the B columns only, so $\lambda = c_B^T B^{-1}$ is uniquely determined. Therefore, there is a **unique** cost vector c with the property that it is 0 at the b entries and satisfies $\bar{c}^T = c^T - \lambda^T A$ for some λ. In the unmodified algorithm the vector of reduced costs is always a vector that satisfies these.

Therefore, we just need to show that the row operation gives such a vector. We know that we get

$$\bar{c_k} - \bar{c_j}\left[\frac{1}{d_l}(y_k)_l\right]$$

We want to show that this satisfies the properties above with B_{new} in place of B.

Since the row vector with k'th component $\bar{c_j}\left[\frac{1}{d_l}(y_k)_l\right]$ is a linear combination of rows of the Tableau interior, it is $v^T B^{-1} A$ for some v so the first property is satisfied. We just need to show the "0 in the basis directions" property then we will be done.

Case 1. k=j

In this case, the new cost is

$$\bar{c_j} - \bar{c_j}\left[\frac{1}{d_l}\left(B^{-1}A_j\right)_l\right] = \bar{c_j} - \bar{c_j}\left[\frac{d_l}{d_l}\right] = 0$$

So we are done

Case 2. The remaining basis variables

Their entry in the old row is 0 in the old tableau and we just need to show it stays that way in the new Tableau. The entry in the "pivot" row is what it was before, ie $\left(B^{-1}A_{B(i)}\right)_l$, which is the l component of the i basis vector which is 0, so 0 will be added to the top row.

This completes the proof that took ALL DAY WHEN I THOUGHT THIS WOULD BE EASY BRUUUUHH

□

Ok now for an example.

Example. Consider the problem of minimizing $-x_1 - x_2 - x_3$ subject to

$$x_1 + 2x_2 + 2x_3 \leq 10$$

$$2x_1 + x_2 + 2x_3 \leq 10$$

$$2x_1 + 2x_2 + x_3 \leq 20$$

$$x_1, x_2, x_3 \geq 0$$

By adding slack variables we can put this into standard form as follows:

$$x_1 + 2x_2 + 2x_3 + x_4 = 10$$

$$2x_1 + x_2 + 2x_3 + x_5 = 10$$

$$2x_1 + 2x_2 + x_3 + x_6 = 20$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

Here,

$$A = \begin{pmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{pmatrix}$$

We will start with $(0, 0, 0, 10, 10, 20)$ as a basic feasible solution. Conveniently, here, $B = I$ so we do not have to do any matrix inverting.

Often if we start with inequality constraints we add slack variables and set only them to be non-zero as above.

To find the vector of reduced costs, use the formula $\bar{c}^T = c^T - c_B^T B^{-1} A$, so

$$\bar{c} = (-1, -1, -1, 0, 0, 0) - (0, 0, 0) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} A = (-1, -1, -1, 0, 0, 0)$$

Now make the table:

$-c_B^T x_B$	$\bar{c_1}$ $\bar{c_2}$ $\bar{c_3}$...$\bar{c_n}$
$x_{B(1)}$	$B^{-1}A_1$,$B^{-1}A_2$,...,$B^{-1}A_n$
$x_{B(2)}$	
$\vdots$	
$x_{B(m)}$	

Filling in the values, noting that $B^{-1}A = A$

0	-1	-1	-1	0	0	0
x_4 :10	1	2	2	1	0	0
x_5 :10	2	1	2	0	1	0
x_6 :20	2	2	1	0	0	1

Now we pick a reduced cost which is negative. We will follow the algorithm as written above the theorem above carefully. We will pick $\bar{c_1}$. Now we set d to $-B^{-1}A_j$ which is $(-1, -2, -2)$, which is not ≥ 0 so we continue. The minimum value of $-\frac{x_{B(i)}}{d_i}$ is 5 (from $B(2)$, as $\frac{10}{2}$ is smaller than $\frac{20}{2}$ and $\frac{10}{1}$) so we set $l = 2$ and we have $\theta^* = 5$. Now instead of $B = (4, 5, 6)$ we will have $B = (4, 1, 6)$ since 1 is our j and $B(2) = 5$ is the one being replaced. We will change the table as follows by doing row operations:

0	-1	-1	-1	0	0	0
x_4 :10	1	2	2	1	0	0
x_1 :5	1	0.5	1	0	0.5	0
x_6 :20	2	2	1	0	0	1

5	0	-0.5	0	0	0.5	0
x_4 :5	0	1.5	1	1	-0.5	0
x_1 :5	1	0.5	1	0	0.5	0
x_6 :10	0	1	-1	0	-1	1

Now $\bar{c_2} < 0$ so we will look at the 2nd basic direction. It turns out that this time, the ratios are

$$\frac{x_{B(1)}}{d_{B(1)}} = \frac{5}{1.5}, \frac{x_{B(2)}}{d_{B(2)}} = \frac{5}{0.5}, \frac{x_{B(3)}}{d_{B(3)}} = \frac{10}{1}$$

Therefore, the "favourite element" is the 1.5 so we take this row which corresponds to B(1) and we change B(1) to j which is 2. We do the following row operations:

Set Row 1 to $\frac{2}{3}R_1$. Add half of row 1 to row 0. Subtract half of row 1 from row 2. Subtract row 1 from row 3. We get

20/3	0	0	1/3	1/3	1/3	0
x_2 :10/3	0	1	2/3	2/3	-1/3	0
x_1 :10/3	1	0	2/3	-1/3	2/3	0
x_6 :20/3	0	0	-5/3	-2/3	-2/3	1

Since the top row (Vector of reduced costs) is non-negative, we deduce that we must be at an optimal solution. Therefore, the optimal solution for this problem is

$$\left(\frac{10}{3}, \frac{10}{3}, 0, 0, 0, \frac{20}{3}\right)$$

For the original problem (since the last 3 variables are slack variables) the optimal solution is just

$$\left(\frac{10}{3}, \frac{10}{3}, 0\right)$$

By the way, here's a funny fact:

In A level decision maths the method to solve linear programs is as follows: Draw the feasible region. Make your ruler parallel to lines of constant f. Identify the furthest point where part of the ruler intersects part of the region. That is it. Lmao. It sounds so funny when I say it.

[Lecture 8 ends]

4 Game theory

Definition. A two person zero sum game is a game where player 1 has m actions they can take and player 2 has n actions they can take. There is an n^*m matrix a with the property that a_{ij} is the value that player 1 wants to maximize and player 2 wants to minimize. For example, 1 would be a win for player 1, and -1 would be a win for player 2. This is called the payoff matrix.

As an example, here is the payoff matrix for rock paper scissors:

	R	P	S
R	0	-1	1
P	1	0	-1
S	-1	1	0

We will first consider the case where player 1 plays, and then player 2 plays, and then the game ends. Here, we expect that player 2 will play whatever minimizes your payoff, so we will play whatever maximizes this minimum.

Ie, P1 will try to solve

$$\max_{i\in\{1,2,\ldots,m\}} \min_{j\in\{1,2,\ldots,n\}} a_{ij}$$

If P2 plays first, they will try to solve the reverse problem, ie

$$\min_{j\in\{1,2,\ldots,n\}} \max_{i\in\{1,2,\ldots,m\}} a_{ij}$$

Example. Suppose $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Here, if we play row 1 then player 2 will play column 1 to minimize our reward, but if we play row 2 player 2 will play column 1 to minimize our reward. Therefore, the best move is to play row 2 and we will get a payout of 3 if player 2 is smart.

Definition. We say that the thing above is a "saddle point" if neither player has a motivation to play differently.

Example. Suppose $A = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}$. Then from Player 1's perspective, if they play row 1 Player 2 would want to play row 2. If they play row 2 player 2 would want to play row 1. From Player 2's perspective, they pick column 1 player 1 would want to play column 1 but if they pick column 2 player 2 would want to play column 2. In this case, it matters who is playing first, unlike in the previous case where either way we would end up on 3 if both players were smart.

Definition. Pure strategy for P1 and P2 is when they pick a fixed action. Mixed strategy is when they pick randomly and pick a probability distribution of how to play.

Ie, P1 picks a distribution P on their actions and P2 picks a distribution Q on their actions.

For player 1, their expected payoff if player 2 tries to hurt them as much as possible is

$$\min_{j\in\{1,2,\ldots,n\}} \sum_i a_{ij}p_i$$

Therefore, player 1 wants to pick p such that they maximize the above quantity.

We can express this a bit differently as follows:

We want to maximize v such that $A^T p \geq ve$ where $e = (1, 1, ..., 1)^T$ (Ie, we want $A_{ij}p_j$ to be greater than v for all i), subject to $p \geq 0$, $e.p = 1$. Ie, it is because we want to maximize the lowest component of $A^T p$.

Similarly, Player 2's problem is to minimize w subject to $A^T q \leq we$ where $e.q = 1$ and $q \geq 0$.

Theorem. The two problems above expressed "a bit differently" are duals of eachother.

Proof. By symmetry it is sufficient to show that Player 2's problem is dual to Player 1's problem.

The Lagrangian of player 2's problem will be

$$L(w, q, s, \lambda_1, \lambda_2) = w + \lambda_1.(Aq + s - we) - \lambda_2(e.q - 1)$$

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

where $s \geq 0$, $(\lambda_1, \lambda_2) \in \Lambda$. To make this not unbounded below, the constraints are:

$$\lambda_1.e = 1$$

$$\lambda_1^T A \geq \lambda_2 e^T$$

$$\lambda_1 \geq 0$$

We want to pick q such that this is maximized subject to being in Λ.

If we interpret λ_1 as p and λ_2 as v this is exactly player 1's problem.

□

We will now determine when a strategy is optimal. Recall that for a general linear program, the optimality conditions were as follows: Primal feasibility, dual feasibility and $(c_j - \lambda.A_j)\, x_j = 0$.

The first 2 conditions are automatic. Note that here, if v, w will attain their maximum then the inequalities $A^T p \geq v.e$ and $A^T q \leq w.e$ will be equalities. Ie, we have $A^T q - we = 0$, in particular $\left(A^T q - we\right)^T p = 0$ and $q^T \left(A^T p - ve\right) = 0$.

Reason the maximum v, w is attained: the set of $A^T p$ and Aq is in a closed set as p lives in a closed set

It follows that $q^T Ap = (we)\,.p$ and that $q^T A^T p = q.\,(ve)$. Note that transposes of a scalar do not change the result

It therefore follows that

$$v\,(q.e) = w\,(p.e) = q^T Ap$$

Since $p.e = q.e = 1$, we have the following:

Proposition. A strategy for both players is optimal if $v = w = q^T Ap$ (Here v and w are the highest we can get subject to $Ap \leq ve$).

Optimal means that neither player has an incentive to change their strategy.

Definition. $q^T Ap$ is called the value of the game if q, p are optimal strategies.

Note that in the case A is a matrix, then by optimality conditions of a linear program, the primal and dual both have the same optimum so by applying that to this case we see that an optimal strategy always exists in the case that a game has a square payoff matrix.

Example. Note that in rock paper scissors, the optimal strategy is to play randomly because if $p = q = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix}$ then the proposition above is satisfied, as we can check:

Given the a from earlier, $q^T Ap = 0$, $Ap = 0e$, $A^T q = 0e$, so $v = w = 0 = q^T Ap$.

[Lecture 9 ends]

To find optimal strategies the procedure is as follows:

1. Look for a saddle point, ie a strategy where neither player would have an incentive to change.

2. Look for dominating actions, ie actions where no matter what the other player plays that are always better - we will see an example of this. If for all j, $A_{i_1 j} \geq A_{i_2 j}$ we should never play i_2 so we can remove that row from the matrix.

3. If this is not sufficient we just use the simplex method.

Example. Suppose the payoff matrix is

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

Since row 1 dominates row 3 we should just look at the matrix

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

Now player 1's problem is to maximise v such that $A^T p \geq ve$. In this case, since there are 2 actions, $p = (p, 1-p)$. Therefore, we have the constraints (1 for each column):

$$2p + 3\,(1-p) \geq v$$

$$3p + (1-p) \geq v$$

$$4p + \frac{1}{2}(1-p) \geq v$$
$$0 \leq p \leq 1$$

Since there is 1 variable, this is actually something which we can solve using the "A level" method. We get that p has to be in some polygon region and we want to maximize what multiple of $(1,1)$ lies in this region. It turns out that this maximum v is $\frac{7}{3}$, attained when $p = \frac{2}{3}$.

For the dual problem (Player 2's strategy) we can use complementary slackness. The tricky part is that $(\lambda_1)_i . s_i = 0$ but the slack variables are implicit. In this case, since the third constraint is not tight, the implicit slack variable will be non-zero so the third component of λ_1 must be 0, ie the third component of q is 0 in the optimum of the dual problem. We can then solve $p^T Aq = \frac{7}{3}$ to find the rest of q.

5 Network flows

5.1 The transport problem

Suppose we have a bunch of nodes and there is a cost to travel between nodes (think airports).

Definition. A directed graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq V \times V$ connecting them.

Definition. Each edge has an associated flow. Assume the number of verices is n, and the flow of the edge from vertex i to vertex j is x_{ij}.

Definition. There is a vector $b \in \mathbb{R}^n$ such that b_i is the total flow entering vertex i. We say vertex i is a source if $b_i > 0$ and a sink if $b_i < 0$.

Definition. There is a cost matrix C where C_{ij} is the cost per unit flow along the edge from i to j.

We may have constraints as follows: There is a matrix $\underline{M}$ (often the zero matrix) and a matrix $\overline{M}$ (like the "capacity") such that we have a constraint $\underline{M}_{ij} \leq x_{ij} \leq \overline{M}_{ij}$

We also have an assumption that $\sum_{i=1}^{n} b_i = 0$, since the total amount of stuff in the system should stay constant.

The optimisation problem for this situation is as follows:

$$\text{minimise} \sum_{(i,j)} c_{ij} x_{ij}$$
$$\text{subject to } b_i + \sum_{(i,j)} x_{ji} = \sum_{(i,j)} x_{ij}$$
$$\underline{M}_{ij} \leq x_{ij} \leq \overline{M}_{ij}$$

This condition is just the definition of b, and we note that it follows from adding up this over every i that $\sum b_i + \sum x_{ji} = \sum x_{ij}$ so $\sum b_i = 0$ since the double sums are the same thing.

A special case of this problem is the transport problem: We have n suppliers ($i \in \{1, 2, ..., n\}$) and m consumers ($j \in \{1, 2, ..., m\}$). Each supplier produces s_i amount of goods. Consumer j wants d_j total amount of goods. Also, we want

$$\sum_{i=1}^{n} s_i = \sum_{j=1}^{m} d_j$$

The cost of moving 1 unit from i to j is C_{ij}, and we note that it only makes sense if $x_{ij} \geq 0$.

The transport problem is

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

$$\text{subject to } \sum_{j=1}^{n} x_{ij} = s_i$$

Think the following image is what is going on (Figure 6):

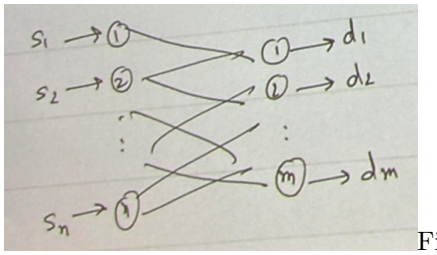

Figure 6

Ie, the total flow from vertex i should be the supply of i, so all the supply is getting consumed. Similarly, we should have

$$\sum_{i=1}^{n} x_{ij} = d_j$$

These conditions together imply$\sum_{i=1}^{n} s_i = \sum_{j=1}^{m} d_j$ as they are both equal to the total sum of x_{ij}.

The issue with trying to apply the simplex method to this is that the assumptions we made when we developed the method are not necessarily true here!

It turns out, a network flow problem can be converted into a transport problem, which we can solve by whatever tools we might develop.

Theorem. Every network flow problem with finite capacities, ie $\overline{M}$ and $\underline{M}$ are finite, can be recast as an equivalent transport problem.

Proof. The idea is we will define things carefully in such a way that we can show any feasible solution to the network flow problem is feasible for this new transport problem we will construct, and the converse is also true.

The first observation is we can assume $\underline{M}_{ij} = 0$ by setting $x'_{ij} = x_{ij} - \underline{M}_{ij}$. We have a new condition $0 \leq x'_{ij} \leq \overline{M}_{ij} - \underline{M}_{ij}$. Now

$$\sum c_{ij} x_{ij} = \sum c_{ij} x'_{ij} + \sum c_{ij} \underline{M}_{ij}$$

and this constant change does not change the minimum, so we will just try to minimize $\sum c_{ij} x'_{ij}$.

We will rewrite the flow condition $b_i + \sum_{(i,j)} x_{ji} = \sum_{(i,j)} x_{ij}$ by setting $b'_i = b_i + \sum_{(i,j)} \underline{M}_{ji} - \sum_{(i,j)} \underline{M}_{ij}$

So we will assume the lower matrix is the zero matrix.

For every vertex i in the network problem, construct a consumer and for every edge (i, j) construct a supplier. Call each supplier (ij) and suppose they are producing $\overline{M}_{ij}$ goods and that consumer i has a demand

$$\sum_{k} \overline{M}_{ik} - b_i$$

Don't worry about why we are doing this for now just take it as the definition. Note that the total supply is equal to the total demand since $\sum b_i = 0$.

We define the cost of transporting from supplier ij to consumer i to be 0 and the cost of transporting from supplier ij to consumer j to be c_{ij}.

Given any feasible flow x_{ij} for the original problem define the flow ij, i and ij, j to be $\overline{M}_{ij} - x_{ij}$ and x_{ij} respectively. From this it follows that the total flow into vertex i is

$$\sum_{k:(i,k)\in E} \left(\overline{M}_{ik} - x_{ik}\right) + \sum_{k:(k,i)\in E} (x_{ki})$$

If this is equal to the demand at i (which it is in the transport problem) we get the equality

$$\sum_{k:(i,k)\in E} \left(\overline{M}_{ik} - x_{ik}\right) + \sum_{k:(k,i)\in E} (x_{ki}) = \sum_{k:(i,k)\in E} \overline{M}_{ik} - b_i$$

This implies that

$$\sum_{k:(i,k)\in E} x_{ik} = b_i + \sum_{k:(k,i)\in E} x_{ki}$$

This is precisely the original condition for the network flow problem. Also, by our construction at the beginning of the proof, we see that $0 \leq x_{ij} \leq \overline{M}_{ij}$ so x_{ij} satisfies the ineuqality constraints.

The point is that any feasible solution to the network flow problem is feasible for this problem, and the converse is also true. This follows since we have considered all the constraints from the original problems.

□

[Lecture 10 ends]

5.2 The transport algorithm

Recall that we want to solve

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

$$\text{subject to } \sum_{j=1}^{m} x_{ij} = s_i$$

$$\sum_{i=1}^{n} x_{ij} = d_j$$

$$x \geq 0$$

Theorem. The optimality conditions are as follows (by reapplying the conditions for a general linear program).

We need x to be feasible, we will have dual variables $\lambda \in \mathbb{R}^n$ corresponding to the n supply constraints, $\mu \in \mathbb{R}^m$ corresponding to the demand, and we will need

$$c_{ij} \geq \lambda_i + \mu_j$$

$$\left(c_{ij} - (\lambda_i + \mu_j)\right) x_{ij} = 0$$

If these hold then x is optimal.

Proof. The lagrangian will be

$$\sum_{(i,j)} c_{ij} x_{ij} - \sum_{i=1}^{n} \lambda_i \left(\sum_{j=1}^{m} x_{ij} - s_i \right) - \sum_{j=1}^{m} \mu_j \left(\sum_{i=1}^{n} x_{ij} - d_j \right)$$

$$= \sum_{i,j} \left(c_{ij} - \lambda_i - \mu_j\right) x_{ij} + \sum_{i=1}^{n} \lambda_i s_i + \sum_{i=1}^{n} \mu_j d_j$$

Therefore to make the minimum not $-\infty$ we need the first condition.

For general linear programs the optimality condition was $\left(c_j - \lambda.A_j\right) x_j = 0$. Here this says $\left(c_{ij} - \lambda.A_{ij}\right) x_{ij} = 0$

This is what the A matrix looks like in this case: it has mn columns for variables and $m + n$ rows for constraints.

I'll show the matrix A for the case $m = 3, n = 4$ to make what I am about to say clear.

$$\left(\begin{array}{cccccccccccc} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \hline \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \end{array}\right)$$

For each fixed i in the s_i constraint it will have a 1 in the row for each value of j where that is the value of i. For each fixed j in the d_j constraint it will have a 1 in the row for each value of i where that is the value of j. This basically means that each column of the matrix will have 2 1's and the rest are 0's, the 2 1's being in the λ_i and μ_j row in the ij column, hopefully this makes sense. Therefore, $\lambda^T A_{(\text{column } ij)}$ indeed becomes $\lambda_i + \mu_j$ so the result follows.

□

Remark. For a given basic feasible solution x, we can assume $\lambda_1 = 0$ because we only care about $\lambda_i + \mu_j$ so we can add $-\lambda_1 (1, 1, ..., 1)$ to λ and subtract that from μ and we will have the same value of $\lambda_i + \mu_j$ for all i, j.

The algorithm has the following table

	μ_1	μ_2	...	μ_m	
λ_1	$\lambda_1 + \mu_1$ x_{11} c_{11}	...	...	$\lambda_1 + \mu_m$ x_{1m} c_{1m}	s_1
λ_2	$\lambda_2 + \mu_1$ x_{21} c_{21}	...	...	$\lambda_2 + \mu_m$ x_{2m} c_{2m}	s_2
...	...	...	...	...	...
λ_n	$\lambda_n + \mu_1$ x_{n1} c_{n1}	...	...	$\lambda_n + \mu_m$ x_{nm} c_{nm}	s_n
	d_1	d_2	...	d_m	

Example. Suppose we have $s_1 = 14$, $s_2 = 10$, $s_3 = 9$ and we have $d_1 = 12$, $d_2 = 5$, $d_3 = 8$, $d_4 = 8$. The total supply (30) is equal to the total demand (30).

The cost matrix is as follows:

$$\left(\begin{array}{cccc} 5 & 3 & 4 & 6 \\ 2 & 7 & 4 & 1 \\ 5 & 6 & 2 & 4 \end{array}\right)$$

We start by coming up with a solution the greedy way: We have s_1 give 12 to d_1, give its remaining 2 to d_2, s_2 gives 3 to d_2 to satisfy d_2 and so on. Therefore our table is as follows:

12 5	2 3	0 4	0 6
0 2	3 7	7 4	0 1
0 5	0 6	1 2	8 4

This has the x and c values but we need to calculate the $\lambda$and μ values. To do this we look at our optimality conditions.

We note that if $x_{ij} > 0$ then we need $c_{ij} = \lambda_i + \mu_j$. For example, we have $\lambda_1 + \mu_1 = 5$. As before we assume $\lambda_1 = 0$, so we get $\mu_1 = 5$. We get more equations:

$\mu_2 = 3$

$\lambda_2 + \mu_2 = 7$

$\lambda_2 + \mu_3 = 4$

$\lambda_3 + \mu_3 = 2$

$\lambda_3 + \mu_4 = 4$

We use these values to fill in the table. We get

5	3	0	2
12	2	0	0
5	3	4	6
9	7	4	6
0	3	7	0
2	7	4	1
7	5	2	4
0	0	1	8
5	6	2	4

In the thing in row 2 column 1 we have $\lambda_i + \mu_j > c_{ij}$ so the solution is not optimal.

Now here is what we do: We can push as much flow as possible into the edge from 2 to 1, but we need to satisfy the total supply total demand condition.

As we see in figure 7, it is like we add that dotted line.

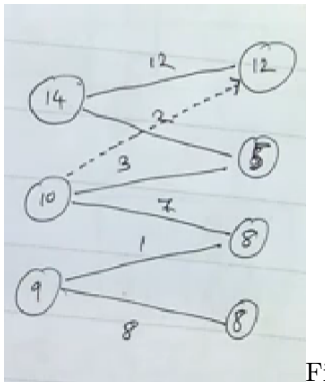

Figure 7

We now have a cycle in the graph: We go supplier 2 to consumer 1 to supplier 1 to consumer 2 back to supplier 2.

What we do now with this cycle is we push as much mass as we can onto the new edge as possible, say δ, then by flow conservation we have to add $-\delta$ to the edge from 1 to 1, δ to the edge from 1 to 2 and $-\delta$ to the edge from 2 to 2, and then we will satisfy the constraints.

We're really interested in $\bar{c}_{ij}$, which satisfies $c_{ij} = \bar{c}_{ij} + \lambda_i + \mu_j$. We note also that the change in the cost is equal to

$$\delta \left(\sum_{\text{forward}} c_{ij} - \sum_{\text{backward}} c_{ij} \right)$$

Thus,

$$\delta \left(\sum_{\text{forward}} \bar{c}_{ij} + \lambda_i + \mu_j - \sum_{\text{backward}} \bar{c}_{ij} + \lambda_i + \mu_j \right)$$

Note that for every edge in the cycle except for the new edge, $\bar{c}_{ij} = 0$, so we will have

$$\delta \left(\bar{c}_{\text{new edge}} + \sum_{\text{forward}} \lambda_i + \mu_j - \sum_{\text{backward}} \lambda_i + \mu_j \right)$$

The last 2 terms will be 0 since each vertex acts exactly once as an entry point and once as an exit point, so they will cancel out and go to 0.

Thus, since $\bar{c}_{\text{new edge}}$ was < 0 the total cost $\sum x_{ij} c_{ij}$ reduces when we do this.

If δ is never 0 we can always push stuff onto the new edge.

This is the idea, however we don't know when it works: We don't know that it won't get stuck slowly decreasing it forever and we don't know that we'll be able to solve the equations at the start uniquely. Now we will analyze the graph theory of this to show why this algorithm works.

Note that if we have a basic feasible solution, then there will be $n+m-1$ edgdes. This is because a basis for the columns of the A matrix from the proof above can be given as follows: The matrix below shows what the basis would be in the case of the example 7*12 matrix A above.

$$\left(\begin{array}{cccccc} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ \hline \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{array} \right)$$

this is indeed a basis because they are clearly linearly independent and given any of the columns from A we can take a column of the "basis" matrix that agrees with it on the bottom or the top, say the $(0, 1, 0, 0, 0, 1, 0)$ which agrees with column 5 of the basis matrix in the top entry, then change the bottom entry by adding column 3 minus column 4 of the basis matrix. Thus we can use the basis matrix to get any column of the A matrix by doing a similar thing.

Therefore the rank of A is $n+m-1$. Thus by definition any basic feasible solution x will have at most $n+m-1$ non-zero entries.

Proposition. The solution we generate at the start by picking greedily has at most $n+m-1$ non-zero entries.

Proof. At each step we keep going until we either exhaust the next supplier or the consumer. This can only happen at most n+m times, and the last time it happens we exhaust the last supplier and last consumer at the same time so in total we only do at most n+m-1 distinct transactions in the initialisation.

□

Note: If a step exhausts a supplier and a consumer simultaneously before the very end, the problem is degenerate. It would not satisfy assumption 3 of linear programs above and while this can happen we will not consider cases like this.

Proposition. The graph we generate in that greedy process has no cycles

Proof. Suppose it did have a cycle. Consider tracing the cycle. Eventually we will have to have supplier a goes to consumer b goes to supplier c wish $c > a$ and then that goes to consumer d with $d < b$. If this never happened, there could not be a cycle, but yet this contradicts how the greedy process operates, which is a contradiction.

□

At this point we have a graph with no cycles, $n+m$ vertices and $n+m-1$ edges.

When we add a new edge, we create at most 1 cycle - if we somehow created 2 by adding a single edge to a graph with no cycles then we would have had a cycle to begin with. When we do the whole shenanigans with the cycle (which works as it must have an even number of elements as it goes left side right side left side right side etc and must end where it started), we get rid of a cycle.

Now recall some properties of linear programs: Each basic feasible solution is an extreme point and so there are finitely many basic feasible solutions. There are also only finitely many graphs with $n + m - 1$ vertices we could end up on.

Therefore, provided we are not in a "degenerate" case the algorithm always terminates as the cost gets strictly smaller each time but the algorithm only ever goes through a finite set of solutions.

[Lecture 11 ends]

6 Max flow min cut

The problem is as follows. Suppose you have 2 nodes (for example airports) and there are multiple ways to get from one to the other. The flow between any 2 airports would have a maximum capacity and we would want to determine the maximum capacity to get from the start node to the end node while satisfying the capacity constraints of each edge.

Because of the constraint that the amount flowing in and out of each node must be 0 everywhere but the source and the sink, we cannot just fill everything with its full capacity. I don't know why people don't write this out because it has to be said because I personally was wondering this for longer than I care to admit.

We can write this as an optimisation problem. Given a graph G with source vertex 1 and sink vertex n. We can phrase this as a linear program as follows:

$$\text{Maximize } \delta$$
$$\text{Subject to} \sum_j x_{1j} = \delta$$
$$\sum_i x_{in} = \delta$$

Ie, the flow out of vertex 1 equals the flow into vertex 1. We have the additional considtion that the flow into i equals the flow out of i, ie

$$\sum_j x_{ij} = \sum_j x_{ji}$$

We put the maximum capacity condition: We have the condition

$$0 \leq x_{ij} \leq c_{ij}$$

Definition. A cut of a graph G is a partition of V (V is the set of vertices and E is the set of edges) into a set S and V\S. The capacity of cut is defined to be $\sum_{i \in S, j \notin S} c_{ij}$. Ie, the total capacity to get from one partition to the other.

We will consider cuts where S contains the source and V\S contains the sink. Note that it is clear that we can never have more flow than the capacity of any individual cut, so we can try to find the cut with minimum capacity. It is not obvious that this actually gives an optimum though, or how to do this efficiently.

Theorem. Let x be a feasible flow with flow value δ. Then for any cut such that vertex 1 belongs to S and vertex n belongs to V\S we must have $\delta \leq C(S)$ where C means the capacity of the cut S.

Proof. Let X, Y be arbitrary subsets of V. Define

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

ie, the overall amount of flow from X to Y.

Now let S be a cut with the property stated in the theorem. Then note that the total flow from 1 to n from 1 to n, which is δ, can be written as

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

This is because for all $i \neq 1$ the quantity $\sum_{j:(i,j) \in E} x_{ij} - \sum_{j:(j,i) \in E} x_{ji}$ is 0 by flow conservation but for $i = 1$ it is δ. Therefore,

$$\delta = f_x(S, V) - f_x(V, S)$$

$$= f_x(S, S) + f_x(S, V \setminus S) - f_x(V \setminus S, S) - f_x(S, S)$$

The reason we can split f like that is easy to see from the definition.

$$= f_x(S, V \setminus S) - f_x(V \setminus S, S)$$

$$\leq f_x(S, V \setminus S)$$

since flow is always positive

$$= C(S)$$

□

Theorem. It turns out that max flow equals min cut over all cuts with 1 in S and n not in S. In other words, suppose δ is an optimal solution, then there exists a cut with capacity equal to δ.

Proof. The proof is hard to understand, I think I finally get it after a long time and will try to explain.

Consider any feasible flow x and a path $1, v_1, v_2, ..., v_{k-1}, n$ where no nodes are visited twice but the path does not need to have all edges forwards and for simplicity set $v_0 = 1, v_k = n$. We say this path is *augmenting* if it has the property that for every i from 1 to k it is the case that $x_{(v_{i-1})(v_i)} < C_{(v_{i-1})(v_i)}$ or, in the event that the path contains a backwards edge, $x_{(v_i)(v_{i-1})} > 0$.

If we have such a path, then the minimum over all backwards positive flows and forwards negative flows is some $\varepsilon > 0$.

Now note that if we have such a path, we can reduce all the backwards flows by ε and increase all the forwards flows by ε. This will leave the flow in and out of every vertex but v_0 and v_k the same but then increase the δ-value that we are trying to maximize, by ε, meaning it was not optimal before.

Thus, an optimal solution has no augmenting paths. It is not easy to see that we won't get stuck in a loop de-augmenting the paths, but it *is* easy to see that there is in fact an optimum and if you think about it that way you realize we will not get stuck de-augmenting the paths.

Now we are in a position to prove the theorem.

Again let x be optimal and let

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

Since there is no augmenting path from 1 to n by optimality, S does not contain n. However, we previously showed that

$$\delta = f_x(S, V \setminus S) - f_x(V \setminus S, S)$$

We will now prove that $f_x(V \setminus S, S) = 0$.

If it is not 0, then there is a node $v \in V \setminus S$ such that there is a flow from vertex v to a vertex u in S. Thus we could add that edge to the augmenting path to u to obtain an augmenting path to v, so v is in S which is a contradiction and implies $f_x(V \setminus S, S) = 0$.

Now it remains to show that $C(S) = f_x(S, V \setminus S)$. If this were not the case, and in fact $f_x(S, V \setminus S)$ was less than the capacity of S, then there would be an edge from some v' in S to some u' in $V \setminus S$ that is not filled, so we could add u' to an augmenting path in v' and we get a similar contradiction.

□

Proposition. If there is no augmenting path we indeed have an optimal solution.

Proof. Suppose for a contradiction that we could add more flow out of vertex 1 somehow. We would have to do this by adding more flow from vertex 1 to another vertex, say vertex a. Then we would have to push flow out of vertex a or put less flow going into vertex a in order to conserve flow, and we will have to do this in a cascading chain reaction until we reach vertex n and get to stop, and for this to be possible there must exist an augmenting path which is a contradiction.

□

Therefore the Ford-Fulkerson algorithm works as follows:

Step 1: Start from a feasible flow, perhaps $x = 0$.

Step 2: Find an augmenting path from 1 to n by checking every path and then send a maximum amount of flow along it.

Repeat until we have an optimal solution.

Example. Consider the following capacities: Figure 8

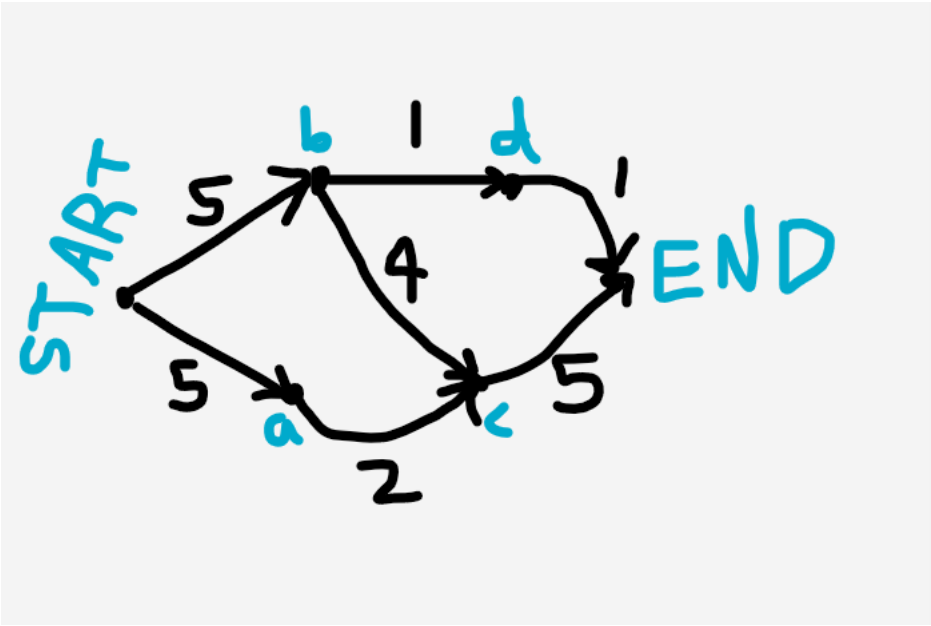

Figure 8

By considering all paths, we can send:

4 from start to b

2 from start to a

3 from b to c

1 from b to d

2 from a to c

5 from c to end

1 from d to end

We now have no augmenting paths so it is optimal.

Alternatively we know it is optimal because the flow is 6 and we can draw a cut such that $S = (\text{START}, a, b, c, d)$ with capacity 6.

Proposition. The algorithm terminates whenever the capacities are all rational.

Proof. By starting at any feasible flow that is all rational and then multiplying appropriately we can assume all capacities are integers. Then at each step the total flow will increase by an integer so in finitely many steps the algorithm will terminate.

□

It is also possible to show that the max flow problem is dual to the min cut problem. We will not prove it here which is fince since we will not use it, it is just an interesting fact.

[Lecture 12 ends]

The actual course ends here. However, the following section is something that is in my opinion easier to understand than what has been here and quite cool and related so I will include it.

7 Dijkstra's algorithm

Suppose we have a situation like in figure 9.

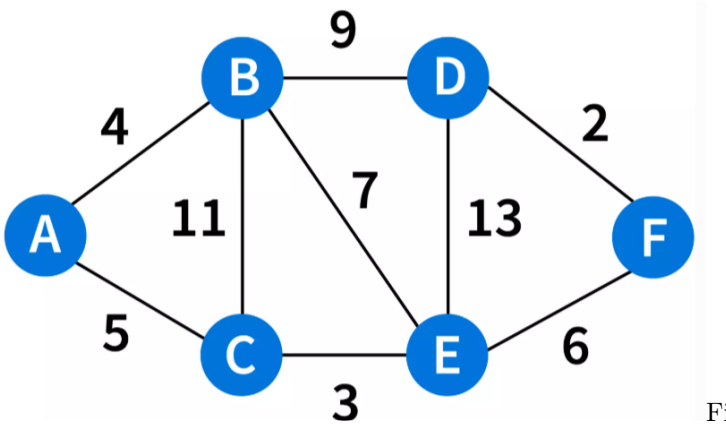

Figure 9

Here the deal is that all edges have weights, say the number of minutes it would take to go along that edge, and we want to find the shortest path from a start point to an end point. Lets say A to F in figure 9 and do that one as an example.

The algorithm works as follows:

From the starting point we find the point that is fastest from it and then we know that we have found the fastest way to get to that point - it cannot possibly be faster to go a slower path and then do some stuff and come back. In the figure 9 context we know it takes 4 minutes to get to B and B is visited - we know the fastest route to get there.

Now we will look at all paths from visited nodes to other nodes. The next fastest path we get as a result is the path from A to C which has distance 5. It is shorter than something like the path ABD which has total distance 13. Therefore we now know we have found all paths of length ≤ 5 so we mark C as visited with the shortest path from A to C having distance 5.

Now we look at the paths ABD, ABE, ACE, as these are all the explored paths with one new step added to an unvisited node. The shortest one is ACE with length 8 so we now know we have explored all paths with length ≤ 8 and so we know the fastest way to get to E.

Now we mark E as 8, C as 5 and B as 4. The new paths we can explore are BD, ED and EF. The one that yields the shortest total distance is BD which gives ABD having total length 13. So we mark D as takes 13 units to reach. Then we can either go to DF for a 15 unit solution or EF for a 14 unit solution.

It then follows that the fastest path is ACEF. This is basically how Dijkstra's algorithm works.