## 8.6 LINEAR PROGRAMMING AND DUALITY

*Linear programming has equality constraints $Ax = b$ and also inequalities $x \geq 0$.* The cost $c_1x_1 + \cdots + c_nx_n$ that we minimize is linear. So the inputs to the problem are $A, b, c$ and the output is (usually) the vector $x^*$ of minimum cost. The vector $y^*$ of Lagrange multipliers solves a highly important *dual* linear program, using $A^{\mathrm{T}}$.

If both problems have optimal vectors, which is normal, then $c^{\mathrm{T}}x^* = b^{\mathrm{T}}y^*$. The "duality gap" is closed. *The minimum of $c^{\mathrm{T}}x$ equals the maximum of $b^{\mathrm{T}}y$.*

You can see why inequality constraints like $x \geq 0$ are needed. The matrix $A$ is rectangular (with $m < n$). Then $Ax_n = 0$ has many solutions, and some of them probably have negative cost. If we could add large multiples of this $x_n$ to any particular solution of $Ax = b$, we could drive the cost to $-\infty$.

In reality, even Enron could not continue to buy negative amounts of energy and pay negative prices. When the components $x_j$ represent purchases, we expect to see $n$ constraints $x_j \geq 0$. They combine into the vector inequality $x \geq 0$.

Here are the statements of the twin linear programs. The $m$ equations $Ax = b$ have $m$ Lagrange multipliers in $y$. Notice how $n$ inequalities $x \geq 0$ in the primal produce $n$ inequalities $A^{\mathrm{T}}y \leq c$ in the dual.

**Primal** Minimize $c^{\mathrm{T}}x$ with constraints $Ax = b$ and $x \geq 0$ (1)

**Dual** Maximize $b^{\mathrm{T}}y$ with constraints $A^{\mathrm{T}}y \leq c$. (2)

One new word. Vectors are called ***feasible*** when they satisfy the constraints. In each problem, the set of all vectors satisfying the constraints is the ***feasible set.*** In general, one or both of the feasible sets could be empty. The primal constraints $x_1 + x_2 = -1$ and $x_1 \geq 0$ and $x_2 \geq 0$ cannot be satisfied by any $x_1$ and $x_2$. We don't expect to see this (an empty feasible set) in good applications.

Let me come immediately to the essence of the theory, before the algorithms. Weak duality is easy, and you will see how it uses the inequality $A^{\mathrm{T}}y \leq c$.

**Weak duality** $b^{\mathrm{T}}y \leq c^{\mathrm{T}}x$ for any feasible $x$ and $y$ (3)

The proof only needs one line, using $Ax = b$ and $x \geq 0$ and $A^{\mathrm{T}}y \leq c$:

**Proof**
$$b^{\mathrm{T}}y = (Ax)^{\mathrm{T}}y = \boldsymbol{x}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}) \leq \boldsymbol{x}^{\mathrm{T}}\boldsymbol{c} = c^{\mathrm{T}}x\,. \tag{4}$$

The step to watch is the inequality. It used both constraints $x \geq 0$ and $A^{\mathrm{T}}y \leq c$. If we only know that $7 \leq 8$, we cannot be sure that $7x \leq 8x$ until we know $x \geq 0$. The inequality $0 \leq x^{\mathrm{T}}(c - A^{\mathrm{T}}y)$ multiplies each $x_j \geq 0$ by a number $s_j \geq 0$. This important number $s_j$ is the "***slack***" in the $j$th inequality $A^{\mathrm{T}}y \leq c$:

$$\boldsymbol{s} = \boldsymbol{c} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} \text{ is the vector of } \textbf{slack variables}. \text{ Always } \boldsymbol{s} \geq \boldsymbol{0}\,. \tag{5}$$

Weak duality $b^{\mathrm{T}}y \leq c^{\mathrm{T}}x$ is simple but valuable. It tells us what must happen for full duality, when the inequality becomes *equality*. Weak duality has $x^{\mathrm{T}}(c - A^{\mathrm{T}}y) \geq 0$. This is $x^{\mathrm{T}}s \geq 0$, for the slack $s = c - A^{\mathrm{T}}y$. ***Full duality has*** $\boldsymbol{x^{\mathrm{T}}s = 0}$.

The components $x_j$ and $s_j$ are never negative, but $x$ and $s$ have a zero dot product $x^{\mathrm{T}}s = 0$ when duality is reached. *For each $j$, either $x_j$ or $s_j$ must be zero.* This ***complementary slackness*** is the key to duality, at the optimal $x^*$ and $y^*$ and $s^*$:

**Optimality $(x^*)^{\mathrm{T}}s^* = 0$** $\quad x_j^* = 0$ or $s_j^* = 0$ for each $j$. (6)

The hard part is to show that this actually happens (if both feasible sets are non-empty). When it does happen, we know that the vectors $x^*$ and $y^*$ and $s^*$ must be optimal. ***The minimum of $c^{\mathrm{T}}x$ matches the maximum of $b^{\mathrm{T}}y$.***

**Duality** $\quad$ Optimal vectors $x^*$ and $y^*$ give $\boldsymbol{c^{\mathrm{T}}x^* = b^{\mathrm{T}}y^*}$. (7)

The practical problem is how to compute $x^*$ and $y^*$. Two algorithms are right now in an intense competition: **simplex methods** against **interior point methods.** The simplex method proves duality by actually constructing $x^*$ and $y^*$ (often quite quickly). This fundamental method is explained first, and executed by a short code.

Interior point methods work **inside** the feasible set. They put a logarithmic barrier at the boundary ($\log x$ makes $x \leq 0$ impossible). They solve an optimality equation $x^{\mathrm{T}}s = \theta$, often by Newton's method. The solution $x^*(\theta)$ moves along the central path (a terrific name) to reach the optimal $x^*$ when $\theta = 0$.

After those general linear programs, this text focuses on two special problems. One asks for **maximum flow** through a graph. You can guess the equality constraint: *Kirchhoff's Current Law.* Then $A$ is the incidence matrix. Its properties lead to the **max flow-min cut theorem**, and its applications include image segmentation.

The other example brings an explosion of new ideas in sparse compression and sparse sensing. Sparsity comes with $\ell^1$ (not $\ell^2$). The key is to combine both norms:

**Sparse solution from the $\ell^1$ penalty** $\quad$ Minimize $\frac{1}{2}\|Ax - b\|_2^2 + \alpha\|x\|_1$ (8)

The term $\|x\|_1 = \sum |x_i|$ is piecewise linear. Its presence drives $x$ to be sparse. But the slope of $|x_i|$ jumps from $-1$ to $1$, and the derivative fails at $x_i = 0$. LASSO will be our introduction to nonlinear optimization (and our farewell too, if this book is to remain finite). The focus stays on **convex optimization**: the best problems.

Here is a skeleton outline of linear and network and nonlinear optimization:

1. The **simplex method**, moving along the edges of the feasible set.

2. The **interior point barrier method**, moving along the inside central path.

3. The sparsifying influence of the $\ell^1$ norm, in **LASSO** and **basis pursuit**.

4. The special problem of **maximum flow-minimum cut**, with applications.

## Corners of the Feasible Set

The solutions to $m$ equations $Ax = b$ lie on a "plane" in $n$ dimensions. (Figure 8.14 has $m = 1$ and $n = 3$.) When $b = 0$ the plane goes through $x = 0$. It is the nullspace of $A$ and the equation is $Ax_{\text{null}} = 0$. Normally we have $b \neq 0$ and the solutions are $x = x_{\text{null}} + x_{\text{part}}$. Then the plane is shifted away from the origin by a particular solution $x_{\text{part}}$ that solves $Ax = b$.

*Linear programming requires* $x \geq 0$. This restricts $x$ to the "northeast corner" of $\mathbf{R}^n$. Coordinate planes like $x_1 = 0$ and $x_2 = 0$ will cut through the solution plane to $Ax = b$. Those cuts are faces of the feasible set where $Ax = b$ and $x \geq 0$. In Figure 8.14, the three coordinate planes $x_1 = 0, x_2 = 0, x_3 = 0$ cut out a triangle.

The triangle has three corners. **We claim that one of those corners is an $x^*$:**

**Corner** $\quad$ ***$Ax = b$ and $x \geq 0$ and $n - m$ components of $x$ are zero.***

One way to find $x^*$ would be to compute all the corners. Then $x^*$ is the one that minimizes the cost $c^{\mathrm{T}}x$. This is a bad way, because for large $m$ and $n$ there are far too many corners. Which $m$ components of $x$ should be nonzero, solving $Ax = b$?

The winner is a corner because the cost $c^{\mathrm{T}}x$ is linear. If this cost increases in one direction, it will decrease in the opposite direction. A straight line graph on an interval is lowest at an endpoint. (If the cost is constant, all points in the interval are winners.) On a triangle, the cost is a minimum on the boundary. On a boundary edge, the minimum is at a corner. **The simplex method finds that corner $x^*$.**

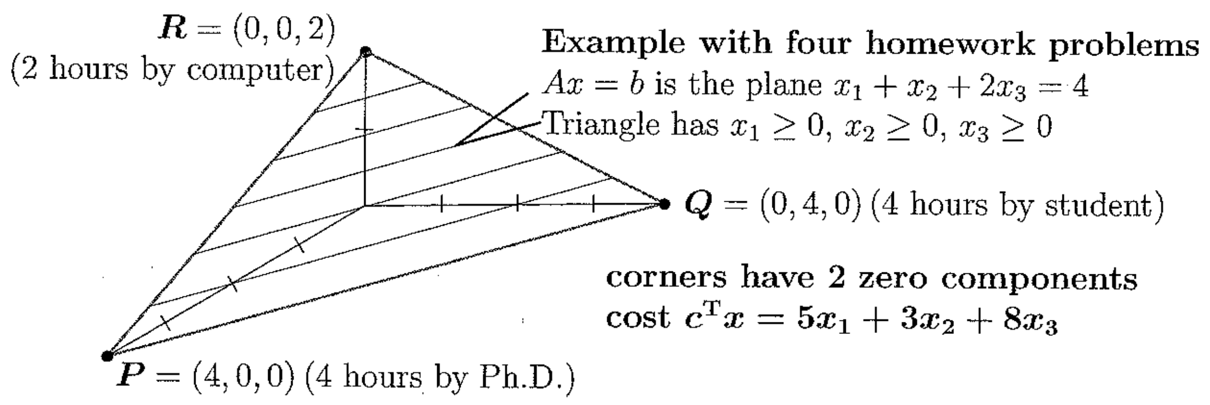
$R = (0, 0, 2)$
(2 hours by computer)
Example with four homework problems
$Ax = b$ is the plane $x_1 + x_2 + 2x_3 = 4$
Triangle has $x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$
$Q = (0, 4, 0)$ (4 hours by student)
corners have 2 zero components
cost $c^{\mathrm{T}}x = 5x_1 + 3x_2 + 8x_3$
$P = (4, 0, 0)$ (4 hours by Ph.D.)

Figure 8.14: A feasible set with corners costing 20, 12, 16. Corner $Q$ is optimal.

The idea is simple. ***Move from corner to corner, decreasing the cost $c^{\mathrm{T}}x$ at every step.*** When you reach a point where all edges going out increase the cost, *stop*. That corner is the optimal $x^*$. Because the number of corners can grow exponentially with $n$ and $m$, this simplex method in the worst case could be slow. In practice the best corner is found quickly. Often $2m$ steps bring us to $x^*$.

## The Simplex Method

Each step travels along an edge from one corner $x_{\text{old}}$ to the next corner $x_{\text{new}}$. One zero component of $x_{\text{old}}$ becomes positive in $x_{\text{new}}$ (the **entering** variable $x_{\text{in}}$). One positive component of $x_{\text{old}}$ becomes zero in $x_{\text{new}}$ (the **leaving** variable $x_{\text{out}}$). The other positive components stay positive as $Ax_{\text{old}} = b$ turns into $Ax_{\text{new}} = b$.

This simplex step solves three $m$ by $m$ systems using the same square matrix $B$. The "basis matrix" $B$ is a submatrix of $A$, with $m$ columns that correspond to nonzeros in $x_{\text{old}}$. At $x_{\text{new}}$, only one column changes from $B_{\text{old}}$ to $B_{\text{new}}$. So we can update $B$ or $B^{-1}$ instead of solving from scratch the new equations that use $B_{\text{new}}$.

May I quickly describe those three systems. The next page shows an example:

1. $\boldsymbol{Bx_{\text{pos}} = b}$ gives the $m$ positive entries at the corner: $x(\text{basis}) = \text{B}\backslash\text{b}$

2. $\boldsymbol{B^{\mathrm{T}}y = c_B}$ decides the entering variable $x_{\text{in}}$ (best edge from $x_{\text{old}}$)

3. $\boldsymbol{Bv = A_{\text{in}}}$ decides the leaving variable $x_{\text{out}}$ (it is zero at the corner $x_{\text{new}}$).

A little more explanation here. The matrix $A$ and cost vector $c$ split into two parts. Those match the positive and zero components of $x$ at the current corner $x_{\text{old}}$:

$$\boldsymbol{Bx_{\text{pos}} = b} \qquad Ax_{\text{old}} = \begin{bmatrix} B & Z \end{bmatrix} \begin{bmatrix} x_{\text{pos}} \\ \mathbf{0} \end{bmatrix} = b \quad \text{and} \quad c = \begin{bmatrix} c_B \\ c_Z \end{bmatrix} \begin{matrix} \text{size } m \\ \text{size } n-m \end{matrix} \tag{9}$$

The vectors $x$ from $Bx_{\text{pos}} = b$ and $y$ from $B^{\mathrm{T}}y = c_B$ achieve $c^{\mathrm{T}}x = c_B^{\mathrm{T}}B^{-1}b = y^{\mathrm{T}}b$. These $x$ and $y$ might look optimal, but the constraint $A^{\mathrm{T}}y \le c$ is probably not satisfied. Everything depends on the slack variables $s = c - A^{\mathrm{T}}y$ being positive:

$$\textbf{Slack is} \begin{bmatrix} 0 \\ \boldsymbol{r} \end{bmatrix} \qquad s = c - A^{\mathrm{T}}y = \begin{bmatrix} c_B \\ c_Z \end{bmatrix} - \begin{bmatrix} B^{\mathrm{T}} \\ Z^{\mathrm{T}} \end{bmatrix} y = \begin{bmatrix} 0 \\ c_Z - Z^{\mathrm{T}}y \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{r} \end{bmatrix} \tag{10}$$

This ***reduced cost vector r*** is the key. If $r \ge 0$, all edges from $x_{\text{old}}$ increase the cost. In that case $s \ge 0$. Then $x_{\text{old}}$ is the optimal $x^*$, and $y$ is the optimal $y^*$.

Until this optimal corner is reached, at least one component of $r$ is negative. The *most negative* component $r_{\text{in}}$ (steepest edge) determines the entering variable $x_{\text{in}}$. Solving $Bv =$ (new column $A_{\text{in}}$) gives the change $v$ in $x_{\text{pos}}$ from one unit of $x_{\text{in}}$.

The first component of $x_{\text{pos}} - \alpha v$ to hit zero identifies the leaving variable $x_{\text{out}}$. The edge has ended. $A_{\text{in}}$ replaces the column $A_{\text{out}}$ in the square basis matrix $B_{\text{new}}$.

That was fast. Every book on linear programming explains the simplex method in more detail (this includes my books on linear algebra, and many others). Equation (12) returns to the reduced costs: $\boldsymbol{r_i} =$ **change in $\boldsymbol{c^{\mathrm{T}}x}$ when $\boldsymbol{x_i}$ moves from 0 to 1.**

## Example of Linear Programming

This example will fit Figure 8.14. The unknowns $x_1, x_2, x_3$ represent hours of work. Those hours are not negative: $x \geq 0$. The costs per hour are \$5 for a Ph.D., \$3 for a student, and \$8 for a computer. (*I apologize for such low pay.*) The Ph.D. and the student get through one homework problem per hour. *The computer solves two problems in one hour.* In principle they can share out the homework, which has four problems to be solved: $x_1 + x_2 + 2x_3 =$ (Ph.D. plus student plus computer) $= 4$.

***Goal: Finish four problems at minimum cost*** $\boldsymbol{c^{\mathrm{T}}x = 5x_1 + 3x_2 + 8x_3}$.

If all three are working, the job takes one hour: $x_1 = x_2 = x_3 = 1$. The cost is $5 + 3 + 8 = 16$. But certainly the Ph.D. should be put out of work by the student (who is just as fast and costs less—this problem is getting realistic). When the student works two hours and the machine works one, the cost is $6 + 8$ and all four problems get solved. We are on the edge $QR$ because the Ph.D. is unemployed: $x_1 = 0$. But the best point on that edge is the corner $Q = x^* = (0, 4, 0)$. **The student solves all four problems in four hours for \$12**—which is the minimum cost $c^{\mathrm{T}}x^*$.

With only one equation in $Ax = x_1 + x_2 + 2x_3 = 4$, the corner $(0, 4, 0)$ has only one nonzero component. When $Ax = b$ has $m$ equations, corners have $m$ nonzeros. The number of possible corners is the number of ways to choose $m$ components out of $n$. This number "$n$ choose $m$" is heavily involved in gambling and probability. With $n = 30$ unknowns and $m = 8$ equations (still small numbers), the feasible set can have $30!/8!\,22!$ corners. That number is $(30)(29)\cdots(23)/8! = 5852925$.

Checking three corners for the minimum cost was fine. Checking five million corners is not the way to go. The simplex method is much faster, as detailed below.

**The Dual Problem** In optimization with constraints, a minimum problem brings a maximum problem. Here is the dual to our example, where $A = [1\ \ 1\ \ 2]$ is transposed. The vectors $b = [4]$ and $c = (5, 3, 8)$ switch between constraint and cost.

**A cheater offers to solve homework problems by looking up the answers.** The charge is $y$ dollars per problem, or $4y$ altogether. (Note how $b = 4$ has gone into the cost.) The cheater can't charge more per problem than the Ph.D. or the student or machine: $\boldsymbol{y \leq 5}$ **and** $\boldsymbol{y \leq 3}$ **and** $\boldsymbol{2y \leq 8}$. This is $A^{\mathrm{T}}y \leq c$. The cheater maximizes the income $4y$.

**Dual Problem** Maximize $y^{\mathrm{T}}b = 4y$ subject to $A^{\mathrm{T}}y \leq c$.

The maximum occurs when $y = 3$. The income is $4y = 12$. The maximum in the dual problem (\$12) equals the minimum in the original problem (\$12). This is duality. Please note that I personally often look up the answers. It's not cheating.

Start with the computer doing all the work: $x_3 > 0$. Then basis $= [3]$ has one index from that one nonzero. Column 3 of $A = [1 \;\; 1 \;\; 2]$ goes into $B = [2]$. Row 3 of the cost $c = [5 \;\; 3 \;\; 8]'$ goes into $c_B = [8]$. A simplex step is quick since $m = 1$:

**1.** $Bx_{\text{pos}} = b = [4]$ gives $x_{\text{pos}} = x_3 = 2$. The computer corner is $x_{\text{old}} = (\mathbf{0}, \mathbf{0}, \mathbf{2})$.

**2.** $B^{\mathrm{T}}y = c_B$ gives $y = 4$. The reduced costs $r$ appear in $c - A^{\mathrm{T}}y = (\mathbf{1}, \mathbf{-1}, \mathbf{0})$. That $-1$ in $r$ identifies the entering variable $x_{\text{in}} = x_2$ (student goes to work).

**3.** $Bv = A_{\text{in}}$ is $2v = 1$. The edge from computer to student has $x_1 = 0$, $x_2 = \alpha$, and $x_3 = 2 - \alpha v$. That edge ends when $\alpha = 2/v = 4$. Then $x_{\text{out}} = x_3$ leaves the basis. This student corner has $\alpha = 4$ units of $x_{\text{in}}$, so $x_{\text{new}} = (\mathbf{0}, \mathbf{4}, \mathbf{0})$.

At the next step, the student corner $x_{\text{new}}$ becomes $x_{\text{old}}$. Then basis $= [2]$ and $B = [1]$ and $x_{\text{pos}} = 4$ is already found (the code only needs step 1 at the starting corner). Step 2 has $c_B = 3$ and $y = 3$ and $c - A^{\mathrm{T}}y = (\mathbf{2}, \mathbf{0}, \mathbf{2})$. *No reduced cost is negative.* So $x^* = (0, 4, 0)$ and $y^* = 3$ are optimal. The student gets the whole job.

---

**Summary** The simplex method stops if $c - A^{\mathrm{T}}y \geq 0$. Otherwise $r_{\text{min}}$ locates the most negative reduced cost and its index in. Then $x_{\text{pos}}$ drops by $\alpha v$ along the edge to $x_{\text{new}}$. The edge ends with $x_{\text{out}} = 0$ when $\alpha = \min(x_i/v_i)$ for $x_i > 0$ and $v_i > 0$.

---

**Different start** The first step may not go immediately to the best $x^*$. The method chooses $x_{\text{in}}$ before it knows how much of that variable to include for $Ax = b$. Starting from $x = (4, 0, 0)$ we could have gone first to $x_{\text{machine}} = (0, 0, 2)$. From there we would have found the optimal $x^* = (0, 4, 0)$.

The more of the entering variable we include, the lower the cost. This has to stop when a positive component of $x$ (which is adjusting to keep $Ax = b$) hits zero. *The* ***leaving variable*** *$x_{\text{out}}$ is the first positive $x_i$ to reach zero,* which signals the new corner. More of $x_{\text{in}}$ would make $x_{\text{out}}$ negative (not allowed). Start again from $x_{\text{new}}$.

When all reduced costs in $r$ are positive, the current corner is the optimal $x^*$. The zeros in $x^*$ cannot become positive without increasing $c^{\mathrm{T}}x$. No new variable should enter: stay with $x_Z = 0$. Here is the algebra that explains reduced costs $r$:

**$Ax = b$ along edge**
$$Ax = \begin{bmatrix} B & Z \end{bmatrix} \begin{bmatrix} x_B \\ x_Z \end{bmatrix} = b. \quad \text{This gives } x_B = B^{-1}b - B^{-1}Zx_Z. \tag{11}$$

The old corner has $x_Z = 0$ and $x_B = B^{-1}b$. At every $x$ the cost is $c^{\mathrm{T}}x$:

**Cost of $x$ along edge**
$$c^{\mathrm{T}}x = \begin{bmatrix} c_B^{\mathrm{T}} & c_Z^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} x_B \\ x_Z \end{bmatrix} = c_B^{\mathrm{T}}B^{-1}b + \left(\boldsymbol{c}_Z^{\mathrm{T}} - \boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{B}^{-1}\boldsymbol{Z}\right)\boldsymbol{x}_Z. \tag{12}$$

The last term $\boldsymbol{r}^{\mathrm{T}}\boldsymbol{x}_Z$ is the direct cost $c_Z^{\mathrm{T}}x_Z$ reduced by the savings from a smaller $x_B$. The constraint (11) updates $x_B$ to maintain $Ax = b$. If $r \geq 0$ we don't want $x_Z$.

## Codes for the Simplex Method

The simplex codes are a gift from Bob Fourer. On the cse website, the code finds a first corner $x \geq 0$ unless the feasible set is empty. In this book, the user inputs $m$ indices in basis to indicate nonzero components of $x$. The code checks $x \geq 0$.

This code starts from the corner that uses the indices in basis. Each step finds a new index in and an old index out. Notice the neat final command basis(out) = in.

```
function [x, y, cost] = simplex(A, b, c, basis)

x = zeros(size(c)) ; v = zeros(size(c)) ;          % column vectors b, c, x, v, y
B = A(:, basis); x(basis) = B\b ; z = .000001       % basis = [m indices ≤ n]
if any (x < -z)                                      % need x ≥ 0 and Ax = b to start
  error ('Bad starting basis, x has component < 0'); end
cost = c(basis)' * x(basis) ;                        % cost c^T x at starting corner
for step = 1:100                                     % take ≤ 100 simplex steps
   y = B'\c(basis) ;                                 % this y may not be feasible (≥ 0)
   [rmin, in] = min(c - A' * y) ;                    % minimum r and its index in
   if rmin > -z                                      % optimality has been reached, r ≥ 0
     break; end                                      % current x, y are optimal x*, y*
   v(basis) = B\A(:, in) ;                           % decrease in x from 1 unit of x_in
[alpha, out] = min(x(basis)./max(v(basis), z)) ;    % alpha locates end x_new of the edge
   if v(basis(out)) < z                              % out = index of first x to reach 0
     error ('cost is unbounded on the set of feasible x'); end
   x(in) = alpha ;                                   % new positive component of x
   cost = cost + x(in) * rmin ;                      % lower cost at end of step
   x(basis) = x(basis) - x(in) * v(basis) ;          % update old x to new corner
   basis(out) = in ;                                 % replace index out by index in
end
```

## Interior Point Methods

The simplex method moves along the edges of the feasible set, eventually reaching the optimal corner $x^*$. ***Interior point methods move inside the feasible set*** (where $x > 0$). These methods hope to go more directly to $x^*$. They work well.

One way to stay inside is to put a barrier at the boundary. Add extra cost as a *logarithm that blows up* when any variable $x_j$ touches zero. The best vector has $x > 0$. The number $\theta$ is a small parameter that we move toward zero.

**Barrier problem** Minimize $c^{\mathrm{T}}x - \theta\,(\log x_1 + \cdots + \log x_n)$ with $Ax = b$ (13)

This cost is nonlinear (but linear programming is already nonlinear from inequalities). The constraints $x_j \geq 0$ are not needed because $\log x_j$ becomes infinite at $x_j = 0$.

The barrier gives an *approximate problem* for each $\theta$. It has $m$ constraints $Ax = b$ with Lagrange multipliers $y_1, \ldots, y_m$. The inequalities $x_i \geq 0$ are hidden inside log $x_i$.

**Lagrangian** $$L(x, y, \theta) = c^{\mathrm{T}}x - \theta\left(\sum \log x_i\right) - y^{\mathrm{T}}(Ax - b) \qquad (14)$$

$\partial L/\partial y = 0$ brings back $Ax = b$. The derivatives $\partial L/\partial x_j$ are interesting!

**Optimality in barrier pbm** $$\frac{\partial L}{\partial x_j} = c_j - \frac{\theta}{x_j} - (A^{\mathrm{T}}y)_j = 0 \quad \text{which is} \quad \boldsymbol{x_j s_j = \theta}. \qquad (15)$$

The true problem has $x_j s_j = 0$, the barrier problem has $x_j s_j = \theta$. The solutions $x^*(\theta)$ lie on the ***central path*** to $x^*(0)$. Those $n$ optimality equations $x_j s_j = \theta$ are nonlinear, and we solve this system iteratively by Newton's method.

The current $x, y, s$ will satisfy $Ax = b, x \geq 0$ and $A^{\mathrm{T}}y + s = c$, *but not* $x_j s_j = \theta$. Newton's method takes a step $\Delta x, \Delta y, \Delta s$. By ignoring the second-order term $\Delta x \Delta s$ in $(x + \Delta x)(s + \Delta s) = \theta$, the corrections in $x, y, s$ come from linear equations:

**Newton step** $$\begin{aligned} A\,\Delta x &= 0 \\ A^{\mathrm{T}}\Delta y + \Delta s &= 0 \\ s_j \Delta x_j + x_j \Delta s_j &= \theta - x_j s_j \end{aligned} \qquad (16)$$

This iteration has quadratic convergence for each $\theta$, and then $\theta$ approaches zero. For any $m$ and $n$, the duality gap $x^{\mathrm{T}}s$ is generally below $10^{-8}$ after 20 to 60 Newton steps. This algorithm is used almost "as is" in commercial interior-point software, for a large class of nonlinear optimization problems. We will show how an $\ell^1$ problem can be made smooth (as Newton wants).

## Barrier Method Example

I will try the same 1 by 3 example, apologizing to the Ph.D. who costs way too much:

$$\text{Minimize } c^{\mathrm{T}}x = 5x_1 + 3x_2 + 8x_3 \text{ with } x_i \geq 0 \text{ and } Ax = x_1 + x_2 + 2x_3 = 4.$$

The constraint has one multiplier $y$. The barrier Lagrangian is $L$:

$$L = (5x_1 + 3x_2 + 8x_3) - \theta\left(\log x_1 + \log x_2 + \log x_3\right) - y\left(x_1 + x_2 + 2x_3 - 4\right). \qquad (17)$$

Seven optimality equations give $x_1, x_2, x_3, s_1, s_2, s_3$, and $y$ (all depending on $\theta$):

| | | |
|---|---|---|
| $\boldsymbol{s = c - A^{\mathrm{T}}y}$ | $s_1 = 5 - y,\ s_2 = 3 - y,\ s_3 = 8 - 2y$ | (18) |
| $\boldsymbol{\partial L/\partial x = 0}$ | $x_1 s_1 = x_2 s_2 = x_3 s_3 = \theta$ | (19) |
| $\boldsymbol{\partial L/\partial y = 0}$ | $x_1 + x_2 + 2x_3 = 4$ | (20) |

Start from the interior point $x_1 = x_2 = x_3 = 1$ and $y = 2$, where $s = (3, 1, 4)$. The updates $\Delta x$ and $\Delta y$ come from $s_j x_j + s_j \Delta x_j + x_j \Delta s_j = \theta$ and $A\Delta x = 0$:

$$\begin{array}{ll} \boldsymbol{3\Delta x_1 + 1\Delta s_1} & 3\Delta x_1 - 1\Delta y = \theta - 3 \\ \boldsymbol{1\Delta x_2 + 1\Delta s_2} & 1\Delta x_2 - 1\Delta y = \theta - 1 \\ \boldsymbol{4\Delta x_3 + 1\Delta s_3} & 4\Delta x_3 - 2\Delta y = \theta - 4 \\ \boldsymbol{A\,\Delta x = 0} & \Delta x_1 + \Delta x_2 + 2\Delta x_3 = 0 \end{array} \qquad (21)$$

Solving for the updates $\Delta x_1, \Delta x_2, \Delta x_3, \Delta y$ gives $x_{\mathsf{new}}$ and $y_{\mathsf{new}}$ (not corners!):

$$\begin{bmatrix} x_{\mathsf{new}} \\ y_{\mathsf{new}} \end{bmatrix} = \begin{bmatrix} x_{\mathsf{old}} \\ y_{\mathsf{old}} \end{bmatrix} + \begin{bmatrix} 3 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 4 & -2 \\ 1 & 1 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \theta-3 \\ \theta-1 \\ \theta-4 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \end{bmatrix} + \frac{\theta}{14} \begin{bmatrix} 1 \\ 3 \\ -2 \\ -11 \end{bmatrix} + \frac{1}{7} \begin{bmatrix} -3 \\ 5 \\ -1 \\ 12 \end{bmatrix}.$$

With $\theta = \frac{4}{3}$ this is $\frac{1}{3}(2,6,2,8)$. We are nearer the solution $x^* = (0,4,0)$, $y^* = 3$.

The reader might notice the **saddle-point matrix $S$** in that equation above. It is ill-conditioned near the $\theta = 0$ solution, where $\mathsf{diag}(S) = c - A^{\mathrm{T}}y^*$ has zeros. Those matrices $S$ just appear everywhere in optimization with constraints.

## Optimization in the $\ell^1$ Norm

Minimizing $Au - b$ in the $\ell^2$ norm leads to $A^{\mathrm{T}}A\widehat{u} = A^{\mathrm{T}}b$ and least squares. Minimizing $Au - b$ in the $\ell^1$ norm leads to linear programming and sparse solutions. Section 2.3 noted the benefits of this sparse alternative (and then stayed in $\ell^2$ to save linearity). Here $m < n$, and this is **basis pursuit** (sparse fit to the data $b$).

The $\ell^1$ norm adds absolute values, so $\|(1,-5)\|_1 = 6$. Those absolute values are the differences between two linear ramps, $x^+ = (1,0)$ and $x^- = (0,5)$:

$$\text{Write} \quad \boldsymbol{x_k = x_k^+ - x_k^-} \quad \text{with} \quad \boldsymbol{x_k^+} = \max(x_k, 0) \geq 0 \quad \text{and} \quad \boldsymbol{x_k^-} = \max(-x_k, 0) \geq 0.$$

Then $|\boldsymbol{x_k}| = \boldsymbol{x_k^+ + x_k^-}$. So the linear program has $2n$ variables $x^+$ and $x^-$:

**Basis pursuit** Minimize $\sum_{1}^{m}(x_k^+ + x_k^-)$ subject to $Ax^+ - Ax^- = b$ (22)

We require $x_k^+ \geq 0$ and $x_k^- \geq 0$. The minimum never has both $x_k^+ > 0$ and $x_k^- > 0$.

**The solution is sparse because it lies at a corner of the feasible set.** $A$ has full row rank, and $x$ has $n-m$ zeros. The system $Ax = b$ has many solutions (it is underdetermined, with too few observations in $b$). The geometry of a polyhedron—*the feasible set*—makes the solution sparse. But we want $x$ to be sparser.

The dual to basis pursuit is neat, involving the $\boldsymbol{\ell^\infty}$ **norm**. The cost vector $c$ is all ones. The constraint matrix $[A \quad -A]$ is transposed according to the dual in (2):

**Dual to basis pursuit** Maximize $b^{\mathrm{T}}y$ with $\begin{bmatrix} A^{\mathrm{T}} \\ -A^{\mathrm{T}} \end{bmatrix} y \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (23)

Every component of $A^{\mathrm{T}}y$ is between 1 and $-1$: *maximum component* $= \|A^{\mathrm{T}}y\|_\infty \leq 1$.

**Dual norms** $\|y\|_{\mathsf{dual}} = \max\limits_{x \neq 0} y^{\mathrm{T}}x/\|x\|_{\mathsf{primal}}$ $\qquad \|x\|_{\mathsf{primal}} = \max\limits_{y \neq 0} y^{\mathrm{T}}x/\|y\|_{\mathsf{dual}}$

When the primal norm is $\|x\|_1$, the dual norm is $\|y\|_\infty$. Example: If $y = (1,3,5)$ choose $x = (0,0,1)$ to find $y^{\mathrm{T}}x/\|x\|_1 = 5$ which is $\|y\|_\infty$. If $x = (1,3,6)$ choose $y = (1,1,1)$ to find $y^{\mathrm{T}}x/\|y\|_\infty = 10$ which is $\|x\|_1$. The $\ell_2$ norm is dual to itself.

## LASSO and More Sparsity

Sparse compression wants fewer than $m$ nonzeros in $x$. So it has to give up on exact solution to the $m$ equations $Ax = b$. We can tighten an $\ell^1$ constraint or increase an $\ell^1$ penalty, to get more zeros. Increase $L$ or reduce $D$ (the names often get mixed):

**Basis Pursuit Denoising** Minimize $\frac{1}{2}\|Ax - b\|_2^2 + L\|x\|_1$ (24)

**LASSO: Least Absolute...** Minimize $\|Ax - b\|_2$ subject to $\|x\|_1 \le D$ (25)

The solution to (24) is also the solution to (25) for some $L$ depending on $D$. LASSO is slightly more useful because the constraint has a dual variable (Lagrange multiplier) that can tell us how much to change $D$. The BPDN solution becomes $x = 0$ (totally sparse!) when the penalty coefficient reaches the value $L^* = \|A^{\mathrm{T}}b\|_\infty$.

Section 4.7 on signal processing worked out an instructive example with increasing $L$. The solution starts from the noiseless basis pursuit (22), with $m$ nonzeros. As $L$ increases, **one of those components moves to zero** (linearly in $L$). Then another component moves to zero (and stays there). By setting $L$ in (24) or $D$ in (25), we control the sparsity of $x$—and more zeros mean less accuracy in $Ax \approx b$. For a noisy signal, we are not concerned about perfect reconstruction of $b$.

May I say: These problems are highly important and the experts are creating fast codes to solve them. The real test is for very large matrices $A$ (a convolution or a randomized sensing matrix, times a wavelet basis matrix). Always there is a competition: Project onto the exact constraints at each step, or move through the interior (but don't slow down for exact Newton steps, that can't win). The cse website will maintain references to both sides, this is remarkable work.

## Convex (Nonlinear) Optimization

Least squares and linear programming are convex problems. So are basis pursuit and LASSO. Practically this whole book has dealt with convex functions. Only the Traveling Salesman will leave this best possible world, and that is what makes his task NP-hard. This is our chance to focus on convexity.

A smooth function $f(x)$ is convex if $d^2f/dx^2 \ge 0$ everywhere. With $n$ variables, $f(x_1, \ldots, x_n)$ is convex if its second-derivative matrix $H$ is **positive semidefinite** everywhere: $H_{ij} = \partial^2 f/\partial x_i \partial x_j$. Graphically, the curve or the surface $y = f(x)$ rises above its tangent lines and tangent planes.

The importance of convexity was well stated by Rockafellar (see [20]): **The great watershed in optimization isn't between linearity and nonlinearity, it is between convexity and nonconvexity.** Here are two equivalent statements of a convex optimization problem, when $f_0(x), \ldots, f_m(x)$ are all convex:

Minimize $f_0(x)$ with $f_i(x) \le b_i$ Minimize $f_0(x)$ in a convex set $K$ (26)

The absolute value function $f(x) = |x|$ and a vector norm $\|x\|$ are also (and importantly) convex, even though they have no derivative at $x = 0$. They pass the test by **staying below the lines connecting points on their graph**:

**Convex function** $\quad f(tx + (1-t)X) \leq t\,f(x) + (1-t)f(X)$ for $0 \leq t \leq 1$. (27)

The right side gives a straight line as $t$ goes from 0 to 1. The left side gives a curve (the graph of $f$). They start equal at $t = 0$ and finish equal at $t = 1$. Straight lines and planes are (barely) convex. Parabolas are (easily) convex if they bend upwards, $\sin x$ is convex between $\pi$ and $2\pi$ but it is **concave** between 0 and $\pi$. It is the *bending* upwards (second derivative) and not the *moving* upwards (first derivative) that makes $f(x)$ a convex function.

It is essential to see that the feasible set $K$ (all $x$ that satisfy the $m$ constraints $f_i(x) \leq b_i$) is a **convex set** in $n$-dimensional space:

**Convex set** $\quad$ The line segment between any points $x$ and $X$ in $K$ stays in $K$.

*Convex functions give convex sets.* If $f_1(x) \leq b_1$ and $f_1(X) \leq b_1$, the right side of (27) stays below $b_1$—so the left side does too. The same is true for each constraint $f_i(x) \leq b_i$. The segment between $x$ and $X$ stays feasible, and $K$ is convex.

*Convex sets $K$ give convex functions.* The **indicator function** $f_K(x)$ is 0 in $K$ and $+\infty$ outside of $K$. The condition "$x$ in $K$" is the same as "$f_K(x) \leq 0$". You see why $f_K$ is a convex function: If $x$ and $X$ are in the convex set $K$, so is the line between them. Then the test (27) is passed.

**Example 1** The intersection of convex sets $K_1, \ldots, K_m$ is a convex set $K$. Especially the intersection of any half-planes $a_i^{\mathrm{T}}x \leq b_i$ is a convex set. This is the feasible set when linear programming (**LP**) requires $Ax \leq b$. Each new constraint just reduces $K$.

**Example 2** The ball $\|x\| \leq 1$ is a convex set in any norm. For the $\ell^2$ norm this is truly a ball. For $\|x\|_1 \leq 1$ we have a diamond, and $\|x\|_\infty \leq 1$ is a box to hold it. The convexity test is the triangle inequality, satisfied by all norms.

**Example 3** The set $K$ of all positive semidefinite matrices is a convex set. If $y^{\mathrm{T}}Ay \geq 0$ and $y^{\mathrm{T}}By \geq 0$ then $y^{\mathrm{T}}(tA + (1-t)B)y \geq 0$. A semidefinite program (**SDP**) minimizes over this set of positive semidefinite matrices in matrix space.

A convex primal minimization leads to a concave dual maximization. The primal-dual pair can be solved by interior point methods—central to the computations. The solution is a saddle point and the optimality equations are KKT conditions—central to the framework of $A^{\mathrm{T}}CA$. And most important, **energy is convex**—central to the applications.

There is a special class of linear programs that have beautiful solutions. By good fortune, those problems have $A$ = *incidence matrix of a graph*. The unknowns in the minimization are potentials $u$ at the nodes. The unknowns $w$ in the dual maximization will be flows on the $m$ edges. The dual problem finds the **maximum flow**.

The flows must satisfy Kirchhoff's Current Law $A^{\mathrm{T}}w = 0$ at every node. Two nodes in the graph are chosen as *the source $s$ and the sink $t$*. The goal is to maximize the total flow from $s$ to $t$. (This flow returns directly to the source as $w_{ts}$ along the dotted line.) Each edge flow $w_{ij}$ is bounded by the **capacity $c_{ij}$** of that edge:

**Maximum flow** Maximize $w_{ts}$ from $t$ to $s$ subject to $A^{\mathrm{T}}w = 0$ and $w \leq c$. (28)

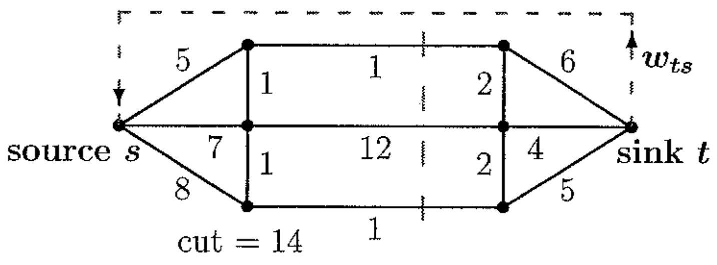

Figure 8.15: Maximize the total flow from $s$ to $t$ with these edge capacities $c_{ij}$.

The best part of this problem is to solve it without linear algebra. The flow across the middle of the graph is at most $1 + 12 + 1 = 14$. You see a **cut** down the middle, and its total capacity is 14. ***All flow must cross this cut.*** *Please look for a cut with capacity less than* 14. Your cut will give a tighter bound on the total flow.

Eventually you will spot the cut of capacity $1 + 2 + 4 + 2 + 1 = 10$. This is the **minimum cut**, and the total flow across the network flow cannot exceed 10. *Does the minimum cut capacity equal the maximum flow?* The answer must be *yes*, or duality wouldn't hold and we would never have mentioned this problem:

**Weak duality** All flows $\leq$ All cuts **Duality** Max flow = Min cut

The minimum cut is the "bottleneck" in the flow graph. You could easily think of applications to an assembly line. The maximum flow fills that bottleneck to capacity.

Away from the bottleneck, the flow pattern for the 10 units is not unique. There are many optimal solutions $w^*$. This degeneracy is not at all unusual in large linear programs (the simplex method can change $w$'s with no improvement). There could also be more than one minimum cut. The point is that we find the minimum quickly.

## Maximum Flow and Bipartite Matching (Marriage)

The classical Ford-Fulkerson proof of duality (*max flow = min cut*) was algorithmic. At every step they looked for an "augmenting path" that would allow additional flow. This is not quite as fast as a greedy algorithm, which never reverses a decision.

We might have to reduce the flow on some edges (send flow backward) to increase the total flow. The cse website links to a family of fast solvers.

Finding the maximum flow also identifies a minimum cut. That cut separates nodes that could still receive more flow from nodes that can't. So this particular cut is full to capacity, proving that max flow equals min cut.

This augmenting path algorithm is *strongly polynomial.* It solves the problem in $O(mn)$ iterations, independent of the capacities $c_{ij}$. Its running time has been steadily lowered, and an alternative "Preflow-Push algorithm" is described in [104].

This special network program includes an even more special problem, when the graph is **bipartite** (two parts). All edges go from one set $S$ of nodes to another set $T$. In the **marriage problem**, an edge goes from a node $i$ in $S$ to $j$ in $T$ when that pair is compatible. All capacities are 1 or 0 (no edge). (At most one marriage, very old-fashioned.) *The maximum flow from $S$ to $T$ is the maximum number of marriages.*

A *perfect matching* connects every node in $S$ along an edge to a node in $T$. The middle figure shows a maximum flow of 3 (4 marriages are impossible here: no perfect matching). To match the flow problem, Figure 8.16 adds a source $s$ and sink $t$.

The adjacency matrix has 1's for all eight edges (compatible pairs). Those 1's can be covered by only 3 lines. Equivalently nodes 2, 3, 4 in $S$ have no edges to 1, 3 in $T$ (zeros in the matrix). This violates Hall's necessary and sufficient condition for a perfect matching: Every $k$ nodes in $S$ must connect to at least $k$ nodes in $T$.

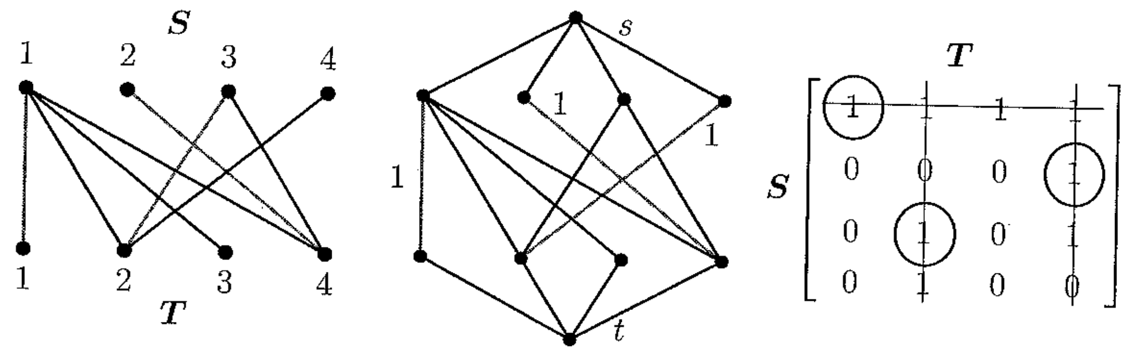

Figure 8.16: This bipartite graph has no perfect matching: *max flow* = *min cut* = 3.

## Network Problems and Traveling Salesmen

The incidence matrix $A$ has two remarkable properties not yet mentioned in this book:

1. Every square submatrix $M$ of $A$ has $\det(M) = 1, -1$ or $0$: **totally unimodular.**

The determinant of $M$ is zero if every row captures both 1 and $-1$ from $A$. If a row captures only one of those, removing that row and column leaves a smaller determinant. Eventually we reach a trivial determinant that yields 1 or $-1$ or 0.

That property tells us that **integer capacities lead to integer maximal flows.** We can require $w_{ij}$ = integer and still have max flow = min cut.

**2.** The corners of the set where $\sum |u_i - u_j| = 1$ have all $u_i = 1$ or $0$ (no fraction).

This property of $Au$, with components $u_i - u_j$ from the incidence matrix $A$, tells us that an optimal corner $u^*$ gives a cut (nodes with $u_i^* = 1$ are on one side). So the integer program for cuts has the same minimum as the linear program for potentials.

Another network problem with a beautiful theory has *edge costs* as the $c_{ij}$:

**Transportation problem** Minimize $\sum\sum c_{ij}\, w_{ij}$ with shipments $w_{ij} \geq 0$ and $Aw = b$. (29)

Those constraints $Aw = b$ are $w_{1j}+\cdots+w_{mj} = b_j$ into market $j$ and $w_{i1}+\cdots+w_{in} = B_i$ from supplier $i$. The cost from supplier to market is $c_{ij}$. The transportation problem has $mn$ unknown flows $w_{ij}$ and $m + n$ equality constraints.

Federal Express solves the dual problem, to know how much it can charge. It collects $u_j$ for each unit delivered to market $j$, and $U_i$ for each unit picked up from supplier $i$. FedEx cannot charge more than the post office: $u_j + U_i \leq c_{ij}$. With those $mn$ constraints they maximize *income* $= \sum u_j b_j + \sum U_i B_i$.

The optimality condition is complementary slackness on each edge: $u_j + U_i = c_{ij}$ or else $w_{ij} = 0$ (no shipment $i$ to $j$). FedEx is cheaper only on unused edges.

This problem also has a fast solution. $A$ is now an incidence matrix with only $+1$'s. Integers still win. But not all network problems have fast solutions!

**Traveling Salesman Problem** Find the cheapest route in a graph that visits each node exactly once.

If the route could be a tree, the problem is easy. A greedy algorithm finds the shortest spanning tree. But the traveling salesman is not allowed to retrace steps. The route must enter and leave each node once. All known algorithms take exponential time (dynamic programming is $n^2 2^n$) which means hours on a supercomputer.

**The traveling salesman problem is NP-hard.** Almost nobody believes that an NP-hard problem can be solved in polynomial time $m^\alpha n^\beta$. Linear programs do have polynomial algorithms (**they are in the class P**). Maximum flows have specially fast times. The outstanding unsolved problem brings a million dollar Clay Prize: **Prove that P $\neq$ NP**: there is no polynomial algorithm for the traveling salesman.

## Continuous Flows

Kirchhoff's Law $A^{\mathrm{T}} w = f$ for a continuous flow is $\operatorname{div} w = \partial w_1/\partial x + \partial w_2/\partial y = f(x,y)$. *The divergence of $w$ is flow out minus flow in.* Transposing, the gradient is like an incidence matrix. Here is **max flow = min cut in a square**, instead of on graphs:

**Flow** Find the flow vector $w(x,y)$ with $\|w\| \leq 1$ and $\operatorname{div} w = C$ that **maximizes** $C$

**Cut** Find the set $S$ in the square that **minimizes** $R =$ (perimeter of $S$)/(area of $S$).

To prove $\boldsymbol{C} \le \boldsymbol{R}$ for all sets $S$, use the Divergence Theorem and $|w \cdot n| \le \|w\| \le 1$:

**Weak duality** $\displaystyle\iint (\operatorname{div} w)\,dx\,dy = \int w \cdot n\,ds \quad C(\text{area of } S) \le (\text{perimeter of } S)$

Minimizing perimeter/area is a classical problem (the Greeks knew that circles win). But hare $S$ lies in the square, partly along its sides. Remove quarter-circles of radius $r$ from the corners. Then calculus finds the best ratio $R = 2+\sqrt{\pi}$ at $r = 1/R$. That $R$ is the "***Cheeger constant***" of the unit square.

It looks easy, but what is the Cheeger constant (minimum ratio of area to volume) inside a cube ?

## Application to Image Segmentation

Suppose you have a noisy image of the heart. Which pixels are part of the heart? Where is the boundary? This is image segmentation and restoration and enhancement. We want and expect a piecewise smooth boundary between pieces of the image, while staying close to the observations $U$.

Assigning pixel values $u_p$ can be expressed as a discrete **energy minimization**:

**Regularized energy sum** $\quad E(u) = E_{\text{data}} + E_{\text{smooth}} = \sum (u_p - U_p)^2 + \sum\sum V(u_p, u_q). \quad (30)$

That last smoothing term will penalize edges that are long or erratic or unnecessary. It is a pixel analog of the Mumford-Shah penalty in continuous segmentation:

**Regularized energy integral** $\quad \displaystyle E(u) = \iint \left[(u-U)^2 + A\,|\nabla u|^2\right] dx\,dy + \textbf{edge length} \quad (31)$

The difficulty is that those penalty terms are ***not convex***. The energy $E(u)$ has many local minima, and pixel space has high dimension. Minimizing $E$ is NP-hard. These problems have many other names: early vision, Bayesian labeling of Markov random fields, Ising models with 2 labels, Potts models with $k$ labels, and more.

In short, **we need good algorithms for impossibly hard problems**. To come within a guaranteed bound is not impossible. We mention the problem here because graph cuts (*not* the normalized cuts of $*2.9$) have been very effective. "Simulated annealing" was too slow. The goal is to allow many pixel values to improve at once, and the correspondence between cuts and pixel labels [21] opened the door to fast max flow-min cut algorithms. I recommend the Graph Cuts Home Page www.cs.cornell.edu/%7Erdz/graphcuts.html.