Advanced algorithms
Freely using the textbooks by Cormen, Leiserson, Rivest, Stein, and of Boyd and Vandenberghe

Péter Gács

Computer Science Department
Boston University

Spring 06
See the course homepage.
In the notes, section numbers and titles generally refer to the book:
**CLSR: Algorithms, second edition.**
For us, a vector is always given by a finite sequence of numbers. Row vectors, column vectors, matrices.

Notation:

- \( \mathbb{Z} \): integers,
- \( \mathbb{Q} \): rationals,
- \( \mathbb{R} \): reals,
- \( \mathbb{C} \): complex numbers.

\( \mathbb{Q}, \mathbb{R}, \mathbb{C} \) are fields (allowing division as well as multiplication). (We may get to see also some other fields later.)

**Addition:** componentwise. Over a field, **multiplication** of a vector by a field element is also defined (componentwise).

**Linear combination.**
Vector space over a field: a set $M$ of vectors closed under linear combination. Elements of the field will be also called scalars.
Examples

- The set $\mathbb{C}$ of complex numbers is a vector space over the field $\mathbb{R}$ of real numbers (2 dimensional, see later).
- It is also a vector space over the complex numbers (1 dimensional).
- $\{(x, y, z) : x + y + z = 0 \}$.
- $\{(2t + u, u, t - u) : t, u \in \mathbb{R} \}$.
Subspace. Generated subspace.

Two equivalent criteria of dependence:

- one of them depends on the others (is in the subspace generated by the others)
- a nontrivial linear combination is 0.

Examples

- \{(1, 2), (3, 6)\}. Two vectors are dependent when one is a scalar multiple of the other.
- \{(1, 0, 1), (0, 1, 0), (1, 1, 1)\}.

Basis in a subspace \(M\): a maximal lin. indep. set.

Theorem

A set is a basis iff it is a minimal generating set.
Examples

- A basis of \( \{(x,y,z) : x + y + z = 0\} \) is \{(0, 1, −1), (1, 0, −1)\}.
- A basis of \( \{(2t + u, u, t − u) : t, y \in \mathbb{R}\} \) is \( (2, 0, 1), (1, 1, −1) \).

Theorem

All bases have the same number of elements.

Proof. Via the exchange lemma.

Dimension of a vector space: this number.

Example

The set of all \( n \)-tuples of real numbers with the property that the sum of their elements is 0 has dimension \( n − 1 \).
Let $M$ be a vector space. If $b_i$ is an $n$-element basis, then each vector $x$ in $M$ in has a unique expression as

$$x = x_1 b_1 + \cdots + x_n b_n.$$ 

The $x_i$ are called the coordinates of $x$ with respect to this basis.

**Example**

If $M$ is the set $\mathbb{R}^n$ of all $n$-tuples of real numbers then the $n$-tuples of form $e_i = (0, \ldots, 1, \ldots, 0)$ (only position $i$ has 1) form a basis. Then $(x_1, \ldots, x_n) = x_1 e_1 + \cdots + x_n e_n$. 
Example

If $A$ is the set of all $n$-tuples whose sum is 1 then the $n-1$ vectors

$$(1, -1, 0, \ldots, 0)$$
$$(0, 1, -1, 0, \ldots, 0)$$
$$\ldots$$
$$(0, 0, 0, 0, \ldots, 0, 1, -1)$$

form a basis of $A$ (prove it!).
Matrices

- \((a_{ij})\). Dimensions. \(m \times n\)
- Unit vector.
- Diagonal matrix \(\text{diag}(a_{11}, \ldots, a_{nn})\)
- Identity matrix.
- Triangular (unit triangular) matrices.
- Permutation matrix.
- Transpose \(A^T\). Symmetric matrix.
Matrix representing a linear map

A $p \times q$ matrix $A$ can represent a linear map $\mathbb{R}^q \rightarrow \mathbb{R}^p$ as follows:

$$
\begin{align*}
  x_1 &= a_{11}y_1 + \cdots + a_{1q}y_q \\
  & \vdots \\
  x_p &= a_{p1}y_1 + \cdots + a_{pq}y_q
\end{align*}
$$

With column vectors $\mathbf{x} = (x_i)$, $\mathbf{y} = (y_j)$ and matrix $A = (a_{ij})$, this can be written as

$$
\mathbf{x} = A\mathbf{y}.
$$

This is taken as the definition of matrix-vector product.

General definition of a linear transformation $F : V \rightarrow W$. Every such transformation can be represented by a matrix, after we fix bases in $V$ and $W$. 
Matrix multiplication

Let us also have

\[
\begin{align*}
y_1 &= b_{11}z_1 + \cdots + b_{1r}z_r \\
\vdots & \quad \vdots \\
y_q &= b_{q1}z_1 + \cdots + b_{qr}z_r
\end{align*}
\]

writeable as \( y = Bz \). Then it can be computed that

\[
x = Cz
\]

where \( C = (c_{ik}) \),

\[
c_{ik} = a_{i1}b_{1k} + \cdots + a_{iq}b_{qk} \quad (i = 1, \ldots, p, \; k = 1, \ldots, r).
\]
We define the matrix product

\[ AB = C \]

from above, which makes sense only for compatible matrices \((p \times q \text{ and } q \times r)\). Then

\[ x = Ay = A(Bz) = Cz = (AB)z. \]

From this we can infer also that matrix multiplication is associative.

**Example**

For \( A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \), \( B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \) we have \( AB \neq BA \).
Transpose of product

Easy to check: \((AB)^T = B^T A^T\).

Inner product

If \(a = (a_i)\), \(b = (b_i)\) are vectors of the same dimension \(n\) taken as column vectors then

\[ a^T b = a_1 b_1 + \cdots + a_n b_n \]

is called their inner product: it is a scalar. The Euclidean norm (length) of a vector \(v\) is defined as

\[ \sqrt{v^T v} = (\sum_i v_i^2)^{1/2}. \]
The (less frequently used) **outer product** makes sense for any two column vectors of dimensions $p, q$, and is the $p \times q$ matrix $ab^T = (a_i b_j)$. 
Example

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}.$$ 

$$(AB)^{-1} = B^{-1}A^{-1}.$$  
$$(A^T)^{-1} = (A^{-1})^T.$$  

A matrix with no inverse is called singular. Nonsingular matrices are also called regular.

Example

The matrix \( \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \) is singular.
Im(A) = set of image vectors of A.
Ker(A) = the set of vectors x with Ax = 0.
Both are subspaces.
Null vector of a matrix: non-0 element of the kernel.

**Theorem**

If \( A : V \to W \) then

\[
\dim \text{Ker}(A) + \dim \text{Im}(A) = \dim(V).
\]

**Theorem**

\( A \) is singular iff \( \text{Ker}A \neq \{0\} \).
The **rank** of a set of vectors: the dimension of the space they generate.

The column rank of a matrix $A$ is $\dim(\text{Im}A)$. (The row rank is harder to interpret.)

**Theorem**

The two ranks are the same (see proof later). Also, $\text{rank}(A)$ is the smallest $r$ such that there is an $m \times r$ matrix $B$ and an $r \times n$ matrix $C$ with $A = BC$.

Interpretation: going through spaces with dimensions $m \to r \to n$.

**Example**

The outer product $A = bc^T$ of two vectors has rank 1, and this product is the decomposition.

**Theorem**

A square matrix is nonsingular iff it has full rank.
A permutation: an invertible map $\sigma : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$. The product of two permutations $\sigma, \tau$ is their consecutive application: $(\sigma \tau)(x) = \sigma(\tau(x))$. A transposition is a permutation that interchanges just two elements.

An inversion in a permutation: a pair of numbers $i < j$ with $\sigma(i) > \sigma(j)$. We denote by Inv($\sigma$) the number of inversions in $\sigma$. A permutation $\sigma$ is even or odd depending on whether Inv($\sigma$) is even or odd.
Proposition

(a) A transposition is always an odd permutation.
(b) \( \text{Inv}(\sigma \tau) \equiv \text{Inv}(\sigma) + \text{Inv}(\tau) \pmod{2} \).
(c) It follows from these that multiplying a permutation with a transposition always changes its parity.
**Definition**

Let \( A = (a_{ij}) \) an \( n \times n \) matrix. Then

\[
\det(A) = \sum_{\sigma} (-1)^{\text{Inv}(\sigma)} a_{1\sigma(1)}a_{2\sigma(2)}\cdots a_{n\sigma(n)}. \tag{1}
\]

**Geometrical interpretation:** the absolute value of the determinant of a matrix \( A \) over \( \mathbb{R} \) with column vectors \( a_1, \ldots, a_n \) is the volume of the parallelepiped spanned by these vectors in \( n \)-space.

**Recursive formula.** Let \( A_{ij} \) be the submatrix (minor) obtained by deleting the \( i \)th row and \( j \)th column. Then

\[
\det(A) = \sum_j (-1)^{i+j} a_{ij} \det(A_{ij}).
\]

We noted that computing \( \det(A) \) using this formula is just as inefficient as using the original definition (1).
Properties:

- $\det A = \det(A^T)$.
- $\det(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ is **multilinear**, that is linear in each argument separately. For example, in the first argument:

$$
\det(\alpha \mathbf{u} + \beta \mathbf{v}, \mathbf{v}_2, \ldots, \mathbf{v}_n) = \alpha \det(\mathbf{u}, \mathbf{v}_2, \ldots, \mathbf{v}_n) + \beta \det(\mathbf{v}, \mathbf{v}_2, \ldots, \mathbf{v}_n).
$$

Hence $\det(0, \mathbf{v}_2, \ldots, \mathbf{v}_n) = 0$.

- **Antisymmetric**: changes sign at the swapping of any two arguments. For example for the first two arguments:

$$
\det(\mathbf{v}_2, \mathbf{v}_1, \ldots, \mathbf{v}_n) = -\det(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n).
$$

Hence $\det(\mathbf{u}, \mathbf{u}, \mathbf{v}_2, \ldots, \mathbf{v}_n) = 0$. 
It follows that any multiple of one row (or column) can be added to another without changing the determinant. From this it follows:

**Theorem**

*A square matrix is singular iff its determinant is 0.*

The following is also known.

**Theorem**

\[ \det(AB) = \det(A)\det(B). \]
Symmetric matrix: an $n \times n$ matrix $A = (a_{ij})$ is symmetric if $a_{ij} = a_{ji}$. To each symmetric matrix, we associate a function $\mathbb{R}^n \to \mathbb{R}$ called a quadratic form and defined by

$$x \mapsto x^T Ax = \sum_{ij} a_{ij} x_i x_j.$$ 

The matrix $A$ is positive definite if $x^T Ax \geq 0$ for all $x$ and we have equality only with $x = 0$. 
For example, if $B$ is a nonsingular matrix then $A = B^T B$ is always positive definite. Indeed,

$$x^T B^T B x = (B x)^T (B x),$$

the squared length of the vector $B x$, and since $B$ is nonsingular, this is 0 only if $x$ is 0.

**Theorem**

A is positive definite iff $A = B^T B$ for some nonsingular $B$. 
It is easy to illustrate here the algebraic divide-and-conquer method. The problem is similar for integers, but is slightly simpler for polynomials.

\[ f = f(x) = \sum_{i=0}^{n-1} a_i x^i, \]
\[ g = f(x) = \sum_{i=0}^{n-1} b_i x^i, \]
\[ f(x)g(x) = h(x) = \sum_{k=0}^{2n-2} c_k x^k, \]
where \( c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0. \)
Let $M(n)$ be the minimal number of multiplications needed to compute the product of two polynomials of length $n$. The school method shows

$$M(n) \leq n^2.$$ 

Can we do better?
For simplicity, assume $n$ is a power of 2 (otherwise, we pick $n' > n$ that is a power of 2). Let $m = n/2$, then

$$f(x) = a_0 + \cdots + a_{m-1}x^{m-1} + x^m(a_m + \cdots + a_{2m-1}x^{m-1}) = f_0(x) + x^m f_1(x).$$

Similarly for $g(x)$. So,

$$fg = f_0g_0 + x^m(f_0g_1 + f_1g_0) + x^{2m}f_1g_1.$$ 

In order to compute $fg$, we need to compute

$$f_0g_0, f_0g_1 + f_1g_0, f_1g_1.$$
How many multiplications does this need? If we compute $f_i g_j$ separately for $i, j = 0, 1$ this would just give the recursion

$$M(2m) \leq 4M(m)$$

which suggests that we really need $n^2$ multiplications.
Trick that saves us a (polynomial) multiplication:

\[ f_0g_1 + f_1g_0 = (f_0 + f_1)(g_0 + g_1) - f_0f_1 - g_0g_1. \quad (2) \]

We found \( M(2m) \leq 3M(m) \). This trick saves us a lot more when we apply it recursively.

\[ M(2^k) \leq 3^kM(1) = 3^k. \]

So, if \( n = 2^k \), then \( k = \log n \),

\[ M(n) < 3^{\log n} = 2^{\log n \cdot \log 3} = n^{\log 3}. \]

\( \log 4 = 2 \), so \( \log 3 < 2 \), so \( n^{\log 3} \) is a smaller power of \( n \) than \( n^2 \).

(It is actually possible to do much better than this.)
Let $L(n)$ be the complexity of multiplication when additions are also counted. The addition of two polynomials of length $n$ takes at most $n$ elementary additions. Taking this into account, the above trick gives the following new estimate:

$$L(2m) \leq 3L(m) + 6m.$$ 

Let us show from here, by induction, that $L(n) = O(n \log^3 n)$.

$$L(2m) \leq 3L(m) + 6m,$$
$$L(4m) \leq 9L(m) + 6m(2 + 3),$$
$$L(8m) \leq 27L(m) + 6m(2^2 + 2 \cdot 3 + 3^2),$$
$$L(2^k) \leq 3^kL(1) + 6(2^{k-1} + 2^{k-2} \cdot 3 + \cdots + 3^{k-1}),$$
$$< 3^k + 2 \cdot 3^k(1 + 2/3 + (2/3)^2 + \cdots) = 3^k(1 + 2 \cdot 3).$$
For matrix multiplication, there is a trick similar to the one seen for polynomial multiplication. Let

\[ A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}, \]

\[ C = AB = \begin{pmatrix} r & s \\ t & u \end{pmatrix}. \]

Then \( r = ae + bg, \ s = af + bh, \ t = ce + dg, \ u = cf + dh. \) The naive way to compute these requires 8 multiplications. We will find a way to compute them using only 7.
Let

\[
P_1 = a(f - h),
\]
\[
P_2 = (a + b)h,
\]
\[
P_3 = (c + d)e,
\]
\[
P_4 = d(g - e),
\]
\[
P_5 = (a + d)(e + h),
\]
\[
P_6 = (b - d)(g + h),
\]
\[
P_7 = (a - c)(e + f).
\]

Then

\[
r = -P_2 + P_4 + P_5 + P_6,
\]
\[
s = P_1 + P_2,
\]
\[
t = P_3 + P_4,
\]
\[
u = P_1 - P_3 + P_5 - P_7.
\]
In all products $P_i$, the elements of $A$ are on the left, and the elements of $B$ on the right. Therefore the calculations leading to (3) do not use commutativity, so they are also valid when $a, b, \cdots, g, h$ are matrices. If $M(n)$ is the number of multiplications needed to multiply $n \times n$ matrices, then this leads (for $n$ a power of 2) to

$$M(n) \leq n^{\log 7}.$$ 

Taking also additions into account:

$$T(2n) \leq 7T(n) + O(n^2).$$

Read Section 1.4 to recall how to prove from here $T(n) = O(n^{\log 7})$. 
Informal treatment first

\[ a_{11}x_1 + \cdots + a_{1n}x_n = b_1, \]
\[ \vdots \]
\[ a_{m1}x_1 + \cdots + a_{mn}x_n = b_m. \]

How many solutions? Undetermined and overdetermined systems.
For simplicity, let us count just multiplications again.

**Jordan elimination**: eliminating first $x_1$, then $x_2$, and so on.

$$n \cdot n \cdot (n + (n - 1) + \cdots) \approx n^3/2.$$  

**Gauss elimination**: eliminating $x_k$ only from equations $k + 1, k + 2, \ldots$. Then solving a triangular set of equations. Elimination:

$$n(n - 1) + (n - 1)(n - 2) + \cdots \approx n^3 / 3.$$  

**Triangular set of equations**:

$$1 + 2 + \cdots + (n - 1) \approx n^2 / 2.$$
Sparsity and fill-in

Example

A sparse system that fills in (from Chvatal):

\[
\begin{align*}
  x_1 + x_2 + x_3 + x_4 + x_5 + x_6 &= 4, \\
  x_1 + 6x_2 &= 5, \\
  x_1 + 6x_3 &= 5, \\
  x_1 + 6x_4 &= 5, \\
  x_1 + 6x_5 &= 5, \\
  x_1 + 6x_6 &= 5.
\end{align*}
\]

Eliminating \(x_1\) fills in everything. There are some guidelines that direct us to eliminate \(x_2\) first, which leads to no such fill-in.
Outcomes of Gaussian elimination
(Possibly changing the order of equations and variables.)

- **Contradiction:** no solution.
- Triangular system with nonzero diagonal: 1 solution.
- Triangular system with \( k \) lines: the solution contains \( n - k \) parameters \( x_{k+1}, \ldots, x_n \).

\[
\begin{align*}
    a_{11}x_1 & + \cdots + a_{1,k+1}x_{k+1} + \cdots + a_{1n}x_n = b_1, \\
    a_{22}x_2 & + \cdots + a_{2,k+1}x_{k+1} + \cdots + a_{2n}x_n = b_2, \\
    \vdots & \\
    a_{kk}x_k & + \cdots + a_{k,k+1}x_{k+1} + \cdots + a_{kn}x_n = b_k,
\end{align*}
\]

where \( a_{11}, \ldots, a_{kk} \neq 0 \).

Then \( \dim \text{Ker}(A) = n - k \), \( \dim \text{Im}(A) = k \).

The operations performed do not change row and column rank, so we find (row rank) = (column rank) = \( k \).
Duality
The original system has no solution if and only if a certain other system has solution. This other system is the one we obtain trying to form a contradiction $0 = 1$ from the original one, via a linear combination with coefficients $y_1, \ldots, y_m$:

\[
\begin{align*}
    a_{11}y_1 + \cdots + a_{m1}y_m &= 0, \\
    \cdots & \hspace{1cm} \cdots \\
    a_{1n}y_1 + \cdots + a_{mn}y_m &= 0, \\
    b_1y_1 + \cdots + b_my_m &= 1.
\end{align*}
\]

Gives an easy way to prove that the system is unsolvable.
Permutation matrix. $PA$ interchanges the rows, $AP$ the columns.

$$PA = LU$$

Using for equation solution:

$$Pb = PAx = LUx.$$  

From here, forward and back substitution.
Computing the LU decomposition

Let us hope first, we do not have to do any permutations.

\[ A = A_0 = \begin{pmatrix} a_{11} & w^T \\ v & A \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}. \]

\[ A_1 = A' - vw^T/a_{11} \] is the Schur's complement of \( A \).

If \( A_1 \) is singular then so is \( A \) (look at row rank).
Now apply LU decomposition to $A_1$, $A_1 = L'U'$:

$$A = \begin{pmatrix} 1 & 0 \\ \nu/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & L'U' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \nu/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix}.$$  

(show it).
Pivoting (see later). Positive definite matrices do not require it (see later).

**Non-recursive code** (tail recursion). Straightforward from the above formula.

**Putting it all in a single matrix**: look at Figure 28.1
Computing LUP decomposition Move a nonzero element $a_{k1}$ into the top left position using a permutation $Q$ then proceed as before.

$$QA = \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix}.$$

Apply LUP decomposition recursively to the Schur complement: $P'(A' - vw^T/a_{k1}) = L'U'$. Define $P = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix}$. 
\[
PA = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} QA 
\]
\[
= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} 
\]
\[
= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} 
\]
\[
= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{pmatrix} 
\]
\[
= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & L'U' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & U' \end{pmatrix} 
\]
\[
= LU. 
\]
Non-recursive organization.

What if there is no pivot in some column? General form:

\[ PAQ = LU, \quad PA = LUQ^{-1}. \]

Using for equation solution:

\[ Pb = PAx = LUQ^{-1}x. \]

Find \( Pb \) by permutation via \( P \), then \( Q^{-1}x \) by forward and backward substitution, then \( x \) by permutation via \( Q \).
Proposition

A triangular matrix with only \( r \) rows (or only \( r \) columns) and all non-0 diagonal elements in those rows, has row rank and column rank \( r \).

Proof is easy.

Proposition

For an \( n \times n \) matrix \( A \), the row rank is the same as the column rank.

Proof. Let \( PAQ = LU \). If \( U \) has only \( r \) rows then \( L \) needs to have only \( r \) columns, and vice versa, so \( L: n \times r \) and \( U: r \times n \). Let us see that \( r \) is the row rank of \( A \). Indeed, \( A \) has a column rank \( r \) since \( U \) maps onto \( \mathbb{R}^r \) and the image of \( L \) is also \( r \)-dimensional. By transposition, the same is true for \( A^T = U^T L^T \), and hence the row rank is the same as the column rank.
Computing the determinant exactly

See the file `exact-Gauss.pdf`, if interested.

- How large is the determinant? (Homework)
- Every intermediate term in the Gaussian elimination is a fraction whose numerator and denominator are some subdeterminants of the original matrix. (Homework) These are not too large. Hence, if we always cancel (using the Euclidean algorithm) our algorithm is polynomial.
- There is a cheaper way than doing complete cancellation (see `exact-Gauss.pdf`).
- Modular computation
When rounding is unavoidable

**Floating point:** $0.235 \cdot 10^5$ (3 digits precision)

**Complete pivoting:** experts generally do not advise it. Considerations of fill-in are typically given preference over considerations of round-off errors, since if the matrix is huge and sparse, we may not be able to carry out the computations at all if there is too much fill-in.
Example

\begin{align*}
0.0001x + y &= 1 \\
0.5x + 0.5y &= 1
\end{align*} \quad (4)

Eliminate \( x \) : 
\(-4,999.5y = -4999.\)
Rounding to 3 significant digits:

\begin{align*}
-5,000y &= -5,000 \\
y &= 1 \\
x &= 0
\end{align*}

True solution: \( y = 0.999899 \), rounds to 1, \( x = 1,0001 \), rounds to 1. We get the true solution by choosing the second equation for pivoting, rather than the first equation.
Forward error analysis: comparing the solution with the true solution.
We can make our solutions look better introducing backward error analysis: showing that our solution solves precisely a system that differs only a little from the original.
Frequently, partial pivoting (choosing the pivot element just in the \( k \)-th column) is sufficient to find a good solution in terms of forward error analysis. However:

**Example**

\[
\begin{align*}
x + 10,000y &= 10,000 \\
0.5x + 0.5y &= 1
\end{align*}
\]

(5)

Choosing the first equation for pivoting seems OK. Eliminate \( x \) from the second eq:

\[-5000.5y = -4,999 \]
\[y = 1 \text{ after rounding} \]
\[x = 0 \]
This is wrong even if we do backward error analysis: every system

\[ a_{11}x + a_{12}y = 10,000 \]
\[ a_{21}x + a_{22}y = 1 \]

satisfied by \( x = 0, y = 1 \) must have \( a_{22} = 1 \).
The problem is that our system is not well scaled. Row scaling and column scaling:

\[ \sum_{ij} r_i a_{ij} s_j x_j = r_i b_i \]

where \( r_i, s_j \) are powers of 10. Equilibration: we can always achieve

\[
0.1 < \max_j |r_i a_{ij} s_j| \leq 1, \\
0.1 < \max_i |r_i a_{ij} s_j| \leq 1.
\]

Example: in (5), let \( r_1 = 10^{-4} \), all other coeffs are 1: We get back (4), which we solve by partial pivoting as before.
Sometimes, like here, there are several ways to scale, and not all are good.

Example

Choose $s_2 = 10^{-4}$, all other coeffs 1:

\[
\begin{align*}
x + y' &= 10,000 \\
0.5x + 0.00005y' &= 1
\end{align*}
\]

(We could have gotten this system to start with. . . .) Eliminate $x$ from the second equation:

\[
\begin{align*}
-0.49995y' &= -4999 \\
y' &= 10000 \text{ after rounding} \\
x &= 0
\end{align*}
\]

so, we again got the bad solution.

Fortunately, such pathological systems are rare in practice.
Computing matrix inverse from an LUP decomposition: solving equations

\[ AX_i = e_i, \quad i = 1, \ldots, n. \]

**Theorem**

*Multiplication is no harder than inversion.*

**Proof.** Let \( D = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix} \). Its inverse is \( D^{-1} = \begin{pmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix} \). \( \square \)
Theorem

Inversion is no harder than multiplication.

Let \( n \) be power of 2. Assume first that \( A \) is symmetric, positive definite, \( A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix} \). Define the Schur complement as

\[
S = D - CB^{-1}C^T.
\]

We will see later that it is positive definite, so it has an inverse. Verify that if

\[
A' = \begin{pmatrix} B^{-1} + B^{-1}C^T S^{-1}CB^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}
\]

then indeed \( AA' = I \) (it is straightforward in this order).
4 multiplications of size $n/2$ matrices

$$CB^{-1}$$

$$(CB^{-1})C^T$$

$$S^{-1}(CB^{-1})$$

$$(CB^{-1})^T(S^{-1}CB^{-1})$$

and 2 inversions and $c \cdot n^2$ additions:

$$I(2n) \leq 2I(n) + 4M(n) + c_1 n^2 = 2I(n) + F(n),$$

$$I(4n) \leq 4I(n) + F(2n) + 2F(n),$$

$$I(2^k) \leq 2^k I(1) + F(2^{k-1}) + 2F(2^{k-2}) + \cdots + 2^{k-1} F(1).$$
Assume $F(n) \leq c_2 n^b$ with $b > 1$. Then

$$F(2^{k-i})2^i \leq c_2 2^{b(k-b)+i} = 2^{bk}2^{-(b-1)i}.$$ 

So,

$$I(2^k) \leq 2^k I(1) + c_2 2^{b(k-1)}(1 + 2^{-(b-1)} + 2^{-2(b-1)} + \cdots) < 2^k + c_2 2^{b(k-1)}/(1 - 2^{-(b-1)}).$$

Inverting an arbitrary matrix: $A^{-1} = (A^T A)^{-1} A^T$. 
Least squares approximation

Least squares

Data: \((x_1, y_1), \ldots, (x_m, y_m)\).

Fitting \(F(x) = c_1f_1(x) + \cdots + c_nf_n(x)\).

It is reasonable to choose \(n\) much smaller than \(m\) (noise).

\[
A = \begin{pmatrix}
  f_1(x_1) & f_2(x_1) & \cdots & f_n(x_m) \\
  f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\
  \vdots & \vdots & \ddots & \vdots \\
  f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m)
\end{pmatrix}.
\]

Equation \(Ac = y\), generally unsolvable in the variable \(c\). We want to minimize the error \(\eta = Ac - y\). Look at the subspace \(V\) of vectors of the form \(Ac\). In \(V\), we want to find \(c\) for which \(Ac\) is closest to \(y\).
Then $Ac$ is the projection of $y$ to $V$, with the property that $Ac - y$ is orthogonal to every vector of the form $Ax$:

$$(Ac - y)^T Ax = 0 \quad \text{for all } x,$$

$$(Ac - y)^T A = 0$$

$A^T (Ac - y) = 0$$

The equation $A^T Ac = A^T y$ is called the normal equation, solvable by LU decomposition.

**Explicit solution:** Assume that $A$ has full column rank, then $A^T A$ is positive definite.

$c = (A^T A)^{-1} A^T y$. Here $(A^T A)^{-1} A^T$ is called the pseudo-inverse of $A$. 
Proposition

The Schur complement is positive definite.

Proof.

\[
(y^T, z^T) \begin{pmatrix} A & B^T \\ B & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = y^T Ay + y^T B^T z + z^T By + z^T Cz
\]

\[
= (y + A^{-1} B^T z)^T A (y + A^{-1} B^T z) + z^T (C - BA^{-1} B^T) z.
\]

For any \( z \) you can choose \( y \) to make the first term 0.
How about solving a system of linear inequalities?

\[ Ax \leq b. \]

We will try to solve a seemingly more general problem:

maximize \( c^T x \)
subject to \( Ax \leq b. \)

This optimization problem is called a linear program. (Not program in the computer programming sense.)
Example

Three voting districts: urban, suburban, rural.
Votes needed: 50,000, 100,000, 25,000.
Issues: build roads, gun control, farm subsidies, gasoline tax.
Votes gained, if you spend $1000 on advertising on any of these issues:

<table>
<thead>
<tr>
<th>adv. spent</th>
<th>policy</th>
<th>urban</th>
<th>suburban</th>
<th>rural</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>build roads</td>
<td>$-2$</td>
<td>$5$</td>
<td>$3$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>gun control</td>
<td>$8$</td>
<td>$2$</td>
<td>$-5$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>farm subsidies</td>
<td>$0$</td>
<td>$0$</td>
<td>$10$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>gasoline tax</td>
<td>$10$</td>
<td>$0$</td>
<td>$-2$</td>
</tr>
<tr>
<td>votes needed</td>
<td>50,000</td>
<td>100,000</td>
<td>25,000</td>
<td></td>
</tr>
</tbody>
</table>

Minimize the advertising budget $(x_1 + \cdots + x_4) \cdot 1000$. 
The linear programming problem:

minimize \( x_1 + x_2 + x_3 + x_4 \)
subject to
\[-2x_1 + 8x_2 + 10x_4 \geq 50,000\]
\[5x_1 + 2x_2 \geq 100,000\]
\[3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25,000\]

Implicit inequalities: \( x_i \geq 0 \).
Two-dimensional example

maximize $x_1 + x_2$
subject to $4x_1 - x_2 \leq 8$
$2x_1 + x_2 \leq 10$
$5x_1 - 2x_2 \geq -2$
$x_1, x_2 \geq 0$

Graphical representation, see book. Convex polyhedron, extremal points.

The simplex algorithm: moving from an extremal point to a nearby one (changing only two inequalities) in such a way that the objective function keeps increasing.
Worry: there may be too many extremal points. For example, the set of $2n$ inequalities

$$0 \leq x_i \leq 1, \quad i = 1, \ldots, n$$

has $2^n$ extremal points.
Standard form

\[
\begin{align*}
\text{maximize} & \quad c^T x \\
\text{subject to} & \quad Ax \leq b \\
& \quad x \geq 0
\end{align*}
\]

Objective function, constraints, nonnegativity constraints, feasible solution, optimal solution, optimal objective value. Unbounded: if the optimal objective value is infinite.

Converting into standard form:

\[
x_j = x_j' - x_j'', \quad \text{subject to } x_j', x_j'' \geq 0.
\]

Handling equality constraints.
Slack form

In the slack form, the only inequality constraints are nonnegativity constraints. For this, we introduce **slack variables** on the left:

\[ x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij}x_j. \]

In this form, they are also called **basic variables**. The objective function does not depend on the basic variables. We denote its value by \( z \).
Example for the slack form notation:

\[
\begin{align*}
z &= 2x_1 - 3x_2 + 3x_3 \\
x_4 &= 7 - x_1 - x_2 + x_3 \\
x_5 &= -7 + x_1 + x_2 - x_3 \\
x_6 &= 4 - x_1 + 2x_2 - 2x_3
\end{align*}
\]

More generally: \( B = \) set of indices of basic variables, \(|B| = m\). \( N = \) set of indices of nonbasic variables, \(|N| = n\), \( B \cup N = \{1, \ldots, m+n\}\). The slack form is given by \((N, B, A, b, c, v)\):

\[
\begin{align*}
z &= v + \sum_{j \in N} c_j x_j \\
x_i &= b_i - \sum_{j \in N} a_{ij} x_j \quad \text{for} \ i \in B.
\end{align*}
\]

Note that these equations are always independent.
(Maximization is counter-intuitive, but the book is wrong.)

maximize \[ d[t] \]
subject to \[ d[v] \leq d[u] + w(u, v) \] for each edge \((u, v)\)
\[ d[s] \geq 0 \]
Maximum flow

Capacity $c(u, v) \geq 0$.

$$\begin{align*}
\text{maximize} & \quad \sum_v f(s, v) \\
\text{subject to} & \quad f(u, v) \leq c(u, v) \\
& \quad f(u, v) = -f(v, u) \\
& \quad \sum_v f(u, v) = 0 \quad \text{for} \ u \in V - \{s, t\}
\end{align*}$$

The matching problem.
Given $m$ workers and $n$ jobs, and a graph connecting each worker with some jobs he is capable of performing. Goal: to connect the maximum number of workers with distinct jobs.
This can be reduced to a maximum flow problem (see homework and book).
**Minimum-cost flow**

Edge cost $a(u, v)$. Send $d$ units of flow from $s$ to $t$ and minimize the total cost

$$\sum_{u,v} a(u, v)f(u, v).$$

**Multicommodity flow**

$k$ different commodities $K_i = (s_i, t_i, d_i)$, where $d_i$ is the demand. The capacities constrain the aggregate flow. There is nothing to optimize: just determine the feasibility.
Games

A **zero-sum two-person game** is played between player 1 and player 2 and defined by an $m \times n$ matrix $A$. We say that if player 1 chooses a **pure strategy** $i \in \{1, \ldots, m\}$ and player 2 chooses pure strategy $j \in \{1, \ldots, n\}$ then there is **payoff**: player 2 pays amount $a_{ij}$ to player 1.

**Example**

$m = n = 2$, pure strategies $\{1, 2\}$ are called “attack left”, “attack right” for player 1 and “defend left”, “defend right” for player 2. The matrix is

$$A = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}. $$

**Mixed strategy**: a probability distribution over pure strategies. $p = (p_1, \ldots, p_m)$ for player 1 and $q = (q_1, \ldots, q_m)$ for player 2. **Expected payoff**: $\sum_{ij} a_{ij} p_i q_j$.
If player 1 knows the mixed strategy $q$ of player 2, he will want to achieve

$$\max_p \sum_i p_i \sum_j a_{ij}q_j = \max_i \sum_j a_{ij}q_j$$

since a pure strategy always achieves the maximum. Player 2 wants to minimize this and can indeed achieve

$$\min_q \max_i \sum_j a_{ij}q_j.$$

This can be rewritten as a linear programming problem:

minimize $t$
subject to $t \geq \sum_j a_{ij}q_j, \quad i = 1, \ldots, m$
$q_j \geq 0, \quad j = 1, \ldots, n$
$\sum_j q_j = 1.$
Slack form.
A **basic solution**: set each nonbasic variable to 0. **Assume** that there is a basic **feasible** solution (see later how to find one).

**Iteration step idea**: try to raise the value of the objective function by changing a nonbasic variable $x_j$ until some basic variable $x_i$ turns 0 (its equality constraint becomes **tight**). Then **exchange** $x_i$ and $x_j$.

**Question**: if this is not possible, are we done?
Linear Programming

The simplex algorithm

Example

\[
\begin{align*}
    z &= 3x_1 + x_2 + 2x_3 \\
    x_4 &= 30 - x_1 - x_2 + 3x_3 \\
    x_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
    x_6 &= 36 - 4x_1 - x_2 - 2x_3
\end{align*}
\]

Since all \( b_i \) are positive, the basic solution is feasible. Increase \( x_1 \) until one of the constraints becomes tight: now, this is \( x_6 \) since \( \frac{b_i}{a_{i1}} \) is minimal for \( i = 6 \). Pivot operation:

\[
x_1 = 9 - x_2/4 - x_3/2 - x_6/4
\]

Here, \( x_1 \) is the entering variable, \( x_6 \) the leaving variable.
Rewrite all other equations, substituting this $x_1$:

\[
\begin{align*}
  z &= 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \\
  x_1 &= 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \\
  x_5 &= 21 - 3\frac{x_2}{4} - 5\frac{x_3}{2} + \frac{x_6}{4} \\
  x_6 &= 6 - 3\frac{x_2}{2} - 4x_3 + \frac{x_6}{2}
\end{align*}
\]

Formal pivot algorithm: no surprise.

**Lemma**

*The slack form is uniquely determined by the set of basic variables.*

**Proof.** Simple, using the uniqueness of linear forms.
When can we not pivot?

- unbounded case
- optimality

The problem of cycling

It can be solved, though you will not encounter it in practice.

Perturbation, or “Bland’s Rule”: choose variable with the smallest index. (No proof here that this terminates.) Geometric meaning: walking around a fixed extremal point, trying different edges on which we can leave it while increasing the objective.
Initial basic feasible solution
Solve the following auxiliary problem, with an additional variable $x_0$:

$$\begin{align*}
\text{minimize} & \quad x_0 \\
\text{subject to} & \quad a_i^T x - x_0 \leq b \quad i = 1, \ldots, m, \\
& \quad x, \quad x_0 \geq 0
\end{align*}$$

If the optimal $x_0$ is 0 then the optimal basic feasible solution is a basic feasible solution to the original problem.
Each pivot step takes $O(mn)$ algebraic operations.

**How many pivot steps?** Can be exponential. Does not occur in practice, where the number of needed iterations is rarely higher than $3 \max(m, n)$. Does not occur on “random” problems, but mathematically random problems are not typical in practice.

**Spielman-Teng:** on a small random perturbation of a linear program (a certain version of) the simplex algorithm terminates in polynomial time (on average).

There exists also a polynomial algorithm for solving linear programs (see later), but it is rarely competitive with the simplex algorithm in practice.
Primal (standard form): maximize $c^T x$ subject to $Ax \leq b$ and $x \geq 0$. Value of the optimum (if feasible): $z^*$. Dual:

$$
\begin{align*}
A^T y & \geq c \\
y^T A & \geq c^T \\
y & \geq 0 \\
y^T & \geq 0 \\
\min b^T y & \quad \min y^T b
\end{align*}
$$

Value of the optimum if feasible: $t^*$.

Proposition (Weak duality)

$z^* \leq t^*$, moreover for every pair of feasible solutions $x, y$ of the primal and dual:

$$c^T x \leq y^T A x \leq y^T b = b^T y. \quad (6)$$
Use of duality. If somebody offers you a feasible solution to the dual, you can use it to upperbound the optimum of the final (and for example decide that it is not worth continuing the simplex iterations).
Interpretation:
- $b_i = \text{the total amount of resource } i \text{ that you have (kinds of workers, land, machines).}$
- $a_{ij} = \text{the amount of resource } i \text{ needed for activity } j.$
- $c_j = \text{the income from a unit of activity } j.$
- $x_j = \text{amount of activity } j.$

$Ax \leq b$ says that you can use only the resources you have.

**Primal problem:** maximize the income $c^T x$ achievable with the given resources.

**Dual problem:** Suppose that you can buy lacking resources and sell unused resources.
Resource $i$ has price $y_i$. Total income:

$$L(x, y) = c^T x + y^T (b - Ax) = (c^T - y^T A)x + y^T b.$$  

Let

$$f(\hat{x}) = \inf_{y \geq 0} L(\hat{x}, y) \leq L(\hat{x}, \hat{y}) \leq \sup_{x \geq 0} L(x, \hat{y}) = g(\hat{y}).$$

Then $f(x) > -\infty$ needs $Ax \leq b$. Hence if the primal is feasible then for the optimal $x^*$ (choosing $y$ to make $y^T (b - Ax^*) = 0$) we have

$$\sup_x f(x) = c^T x^* = z^*.$$  

Similarly $g(y) < \infty$ needs $c^T \leq y^T A$, hence if the dual is feasible then we have

$$z^* \leq \inf_y g(y) = (y^*)^T b = t^*.$$
Complementary slackness conditions:

\[ y^T(b - Ax) = 0, \quad (y^T A - c^T)x = 0. \]

Proposition

*Equality of the primal and dual optima implies complementary slackness.*

Interpretation:

- Inactive constraints have shadow price \( y_i = 0 \).
- Activities that do not yield the income required by shadow prices have level \( x_j = 0 \).
**Theorem (Strong duality)**

*The primal problem has an optimum if and only if the dual is feasible, and we have*

\[ z^* = \max c^T x = \min y^T b = t^*. \]

This surprising theorem says that there is a set of prices (called **shadow prices**) which will force you to use your resources optimally.

Many interesting uses and interpretations, and many proofs.
Our proof of strong duality uses the following result of the analysis of the simplex algorithm.

**Theorem**

*If there is an optimum \( v \) then there is a basis \( B \subset \{1, \ldots, m + n\} \) belonging to a basic feasible solution, and coefficients \( \tilde{c}_i \leq 0 \) such that*

\[
    c^T x = v + \tilde{c}^T x,
\]

*where \( \tilde{c}_i = 0 \) for \( i \in B \).*

Define the nonnegative variables

\[
    \tilde{y}_i = -\tilde{c}_{n+i} \quad i = 1, \ldots, m.
\]
For any $\mathbf{x}$, the following transformation holds, where $i = 1, \ldots, m$, $j = 1, \ldots, n$:

$$
\sum_j c_j x_j = v + \sum_j \tilde{c}_j x_j + \sum_i \tilde{c}_{n+i} x_{n+i}
$$

$$
= v + \sum_j \tilde{c}_j x_j + \sum_i (-\tilde{y}_i)(b_i - \sum_j a_{ij} x_j)
$$

$$
= v - \sum_i b_i \tilde{y}_i + \sum_j (\tilde{c}_j + \sum_i a_{ij} \tilde{y}_i) x_j.
$$

This is an identity for $\mathbf{x}$, so $v = \sum_i b_i \tilde{y}_i$, and also $c_j = \tilde{c}_j + \sum_i a_{ij} \tilde{y}_i$. Optimality implies $\tilde{c}_j \leq 0$, which implies that $\tilde{\mathbf{y}}_i$ is a feasible solution of the dual.
Any feasible solution of the set of inequalities

\[
Ax \leq b \\
A^T y \geq c \\
c^T x - b^T y = 0 \\
x, \quad y \geq 0
\]

gives an optimal solution to the original linear programming problem.
Theorem (Farkas Lemma, not as in the book)

A set of inequalities $Ax \leq b$ is unsolvable if and only if a positive linear combination gives a contradiction: there is a solution $y \geq 0$ to the inequalities

\[
y^T A = 0,
\]
\[
y^T b < 0.
\]

For proof, translate the problem to finding an initial feasible solution to standard linear programming.
We use the homework allowing variables without nonnegativity constraints:

\[
\begin{align*}
\text{maximize} & \quad z \\
\text{subject to} & \quad Ax + z \cdot e \leq b \\
\end{align*}
\]

Here, \( e \) is the vector consisting of all 1’s. The dual is

\[
\begin{align*}
\text{minimize} & \quad y^T b \\
\text{subject to} & \quad y^T A = 0 \\
& \quad y^T e = 1 \\
& \quad y^T \geq 0
\end{align*}
\]

The original problem has no feasible solution if and only if \( \max z < 0 \) in (7). In this case, \( \min y^T b < 0 \) in (8). (Condition \( y^T e = 1 \) is not needed.)
Primal, with dual variables written in parentheses at end of lines:

\[
\begin{align*}
\text{minimize} & \quad t \\
\text{subject to} & \quad t - \sum_j a_{ij}q_j \geq 0 \quad i = 1, \ldots, m \quad (p_i) \\
& \quad \sum_j q_j = 1, \quad (z) \\
& \quad q_j \geq 0, \quad j = 1, \ldots, n \\
\end{align*}
\]

Dual:

\[
\begin{align*}
\text{maximize} & \quad z \\
\text{subject to} & \quad \sum_i p_i = 1, \\
& \quad -\sum_i a_{ij}p_i + z \leq 0, \quad j = 1, \ldots, n \\
& \quad p_i \geq 0 \quad i = 1, \ldots, m.
\end{align*}
\]
maximize \[ \sum_{v \in V} f(s, v) \]
subject to \[ f(u, v) \leq c(u, v), \quad u, v \in V, \]
\[ f(u, v) = -f(v, u), \quad u, v \in V, \]
\[ \sum_{v \in V} f(u, v) = 0, \quad u \in V \setminus \{s, t\}. \]

Two variables associated with each edge, \( f(u, v) \) and \( f(v, u) \).
Simplify. Order the points arbitrarily, but starting with \( s \) and ending with \( t \). Leave \( f(u, v) \) when \( u < v \): whenever \( f(v, u) \) appears with \( u < v \), replace with \(-f(u, v)\).
maximize $\sum_{v > s} f(s, v)$
subject to

$f(u, v) \leq c(u, v), \quad u < v,$
$-f(u, v) \leq c(v, u), \quad u < v,$

$\sum_{v > u} f(u, v) - \sum_{v < u} f(v, u) = 0, \quad u \in V \setminus \{s, t\}.$

Some constraints disappeared but others appeared, since in case of $u < v$ the constraint $f(v, u) \leq c(v, u)$ is written now $-f(u, v) \leq c(u, v)$.

A dual variable for each constraint. For $f(u, v) \leq c(u, v)$, call it $y^+(u, v)$, for $-f(u, v) \leq c(u, v)$, call it $y^-(y, v)$. For

$\sum_{v > u} f(u, v) - \sum_{v < u} f(v, u) = 0$

call it $y(u)$. 
Dual constraint for each primal variable $f(u, v)$, $u < v$. Since $f(u, v)$ is not restricted by sign, the dual constraint is an equation. If $u, v \neq s$ then $f(u, v)$ has coefficient 0 in the objective function. Let

$$y(u, v) = y^+(u, v) - y^-(u, v).$$

The equation for $u \neq s$, $v \neq t$ is $y^+(u, v) - y^-(u, v) + y(u) - y(v) = 0$, or

$$y(u, v) = y(v) - y(u).$$

For $u = s$, $v \neq t$: $y^+(s, v) - y^-(s, v) - y(v) = 1$, or

$$y(s, v) = y(v) - (-1).$$

For $u \neq s$ but $v = t$, $y^+(u, t) - y^-(u, t) + y(u) = 0$, or

$$y(u, t) = 0 - y(u).$$
For $u = s, v = t$: $y^+(s, t) - y^-(s, t) = 1$, or

$$y(s, t) = 0 - (-1).$$

Setting $y(s) = -1, y(t) = 0$, all these equations can be summarized in $y(u, v) = y(v) - y(u)$ for all $u, v$.

The objective function is $\sum_{u, v} c(u, v)(y^+(u, v) + y^-(u, v))$.

The maximum of any $x^+ + x^-$ subject to $x^+, x^- \geq 0, x^+ - x^- = a$ is $|a|$, so the objective function can be simplified to $\sum_{u, v} c(u, v)|y(u, v)|$. Simplified dual problem:

$$\text{minimize } \sum_{u < v} c(u, v)|y(v) - y(u)|$$
$$\text{subject to } y(s) = -1, \ y(t) = 0.$$ 

Let us require $y(s) = 0, y(t) = 1$ instead; the problem remains the same.
Claim

There is an optimal solution in which each $y(u)$ is 0 or 1.

Proof. Assume that there is an $y(u)$ that is not 0 or 1. If it is outside the interval $[0, 1]$ then moving it towards this interval decreases the objective function, so assume they are all inside. If there are some variables $y(u)$ inside this interval then move them all by the same amount either up or down until one of them hits 0 or 1. One of these two possible moves will not increase the objective function. Repeat these actions until each $y(u)$ is 0 or 1.
Let $y$ be an optimal solution in which each $y(u)$ is either 0 or 1. Let

$$S = \{ u : y(u) = 0 \}, \quad T = \{ u : y(u) = 1 \}. $$

Then $s \in S$, $t \in T$. The objective function is

$$\sum_{u \in S, v \in T} c(u, v).$$

This is the value of the “cut” $(S, T)$. So the dual problem is about finding a minimum cut, and the duality theorem implies the max-flow/min-cut theorem.
Bipartite graph with left set $A$, right set $B$ and edges $E \subseteq A \times B$. Interpretation: elements of $A$ are workers, elements of $B$ are jobs. $(a, b) \in E$ means that worker $a$ has the skill to perform job $b$. Two edges are disjoint if both of their endpoints differ. Matching: a set $M$ of disjoint edges. Maximum matching: a maximum-size assignment of workers to jobs.

Covering set $C \subseteq A \cup B$: a set with the property that for each edge $(a, b) \in E$ we have $a \in C$ or $b \in C$.

Clearly, the size of each matching is $\leq$ the size of each covering set.

**Theorem**

*The size of a maximum matching is equal to the size of a minimum covering set.*

There is a proof by reduction to the flow problem and using the max-flow min-cut theorem.
There are many optimization problems that are not linear programming problems. Still, many of these have a property, convexity, allowing efficient solutions.

**Convex linear combination:** $\sum_i \lambda_i u_i$, where $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$.

A set $S \in \mathbb{R}^n$ is **convex** if the convex linear combination of any two points of $S$ is in $S$. Equivalent definition: if the convex linear combination of any number of points of $S$ is in $S$.

**Proposition**

*The intersection of any number of convex sets is convex.*
A set $S$ is **open** if whenever for every $x \in S$ there is an $\epsilon > 0$ such that for every $y$ with $|y - x| < \epsilon$ we have $y \in S$. That is, with every point $x$ in $S$, a whole neighborhood of $x$ is in $S$ as well. A set $S$ is **closed** if its complement is open.

### Examples

- **subspace**, **affine set** (shifted subspace).
- **hyperplane**: $\{ x : a^T x = b \}$.
- **closed halfspace**: $\{ x : a^T x \leq b \}$.
- **open halfspace**: $\{ x : a^T x < b \}$.
- **closed convex polyhedron**: $\{ x : Ax \leq b \}$.
- **open ball** with center $c$ and radius $r$: $\{ x : |x - c| < r \}$.
- **ellipsoid** $\{ x : x^T A x < r^2 \}$, for a positive definite matrix $A$.
- $\{ X : X$ is a positive definite $n \times n$ matrix $\}$
Theorem (Separating hyperplane)

If $S$ is a convex closed set and $v \notin S$ then there are $a, b$ with $a^T x < b$ for all $x \in S$ and $a^T v > b$.

We will not prove the theorem, but will use it.
Let $L$ be the bounded convex set with extremal points $u_1, \ldots, u_m$ in an $n$-dimensional space. (By a homework) points of $L$ are the convex linear combinations of points $u_1, \ldots, u_m$. Thus, a point $v$ is in $L$ if the following set of inequalities has a solution:

$$\sum_i u_i y_i = v, \quad \sum_j y_j = 1, \quad y \geq 0.$$  

By the separating hyperplane theorem, if $v$ is not in $L$ then there is a separating hyperplane between $L$ and $v$: namely, the following set of inequalities has a solution $x, z$:

$$u_i^T x < z, \quad i = 1, \ldots, m,$$

$$v^T x > z.$$  

This statement, the linear programming duality theorem and the Farkas Lemma are closely related (can also be derived from each other).
Let $S \subseteq \mathbb{R}^n$ be a convex set. A function $f : S \to \mathbb{R}$ is **convex** if the set above its graph, namely

$$\{(x, y) : x \in S, y \geq f(x)\}$$

is convex. Equivalently, if for every $0 < \lambda < 1$ and $u, v$ we have

$$f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v).$$

Same for convex linear combination of several points.

**Examples**

- A (not necessarily homogeneous) linear function $a^T x + b$.
- A *quadratic function* $x^T A x + b^T x + c$ with nonnegative definite $A$.
- $f(x) = 1/x$ for $x > 0$. Here $S = \{x \in \mathbb{R} : x > 0\}$.

A function $f$ is **concave** if $-f$ is convex.
**Extending outside the domain:** For a convex function $f : S \rightarrow \mathbb{R}$, we will define $f(x) = \infty$ for all $x \not\in S$. This is convenient.

**Minimization:** Convex functions are attractive to minimize since for a convex function, a local minimum is also a global minimum (homework problem). Similarly, concave functions are attractive to maximize.
Proposition

- Any convex linear combination of convex functions is convex.
- The maximum (even the supremum) of any number of convex functions is convex.

Example

From the earlier game theory example: the function

\[ f(q) = \max_i \sum_j a_{ij}q_j \]

is convex, so Player 2 is trying to minimize a convex function.

Proposition

If \( f \) is convex then \( x \mapsto f(Ax + b) \) is convex, too.
Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function. Its derivative is a vector function

$$g : \mathbb{R}^n \to \mathbb{R}^n.$$

We write $\nabla f(x) = g(x)$. It is defined by

$$(g(x))_i = \nabla f(x)_i = \frac{\partial f(x)}{\partial x_i}.$$

**Example**

Suppose $x = (x_1, x_2)^T$ and $f(x) = x_1^2/x_2$. Then

$$\nabla f(x) = \begin{pmatrix} 2x_1/x_2 \\ -x_2^2/x_2^2 \end{pmatrix}.$$
The function \( \hat{f}(z) \) defined by

\[
\hat{f}(x + h) = f(x) + \nabla f(x)^T h
\]

is linear and approximates \( f(z) \) well near the point \( z = x \) (that is, \( h = 0 \), this is a first-order approximation in \( h \)). Its graph is the tangent hyperplane of the graph of the function \( f \) in the point \( x \). 

\( f \) is convex iff it is always above its tangent:

\[
f(x + h) \geq f(x) + \nabla f(x)^T h.
\]
The second-order partial derivatives of $f$ form a **matrix**:

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \end{pmatrix}.$$ 

**Example**

Suppose $x = (x_1, x_2)^T$ and $f(x) = x_1^2 / x_2$. Then

$$\nabla f(x) = \begin{pmatrix} 2x_1 / x_2 \\ -x_1^2 / x_2^2 \end{pmatrix}, \quad \nabla^2 f(x) = \begin{pmatrix} 2/x_2 & -2x_1/x_2^2 \\ -2x_1/x_2^2 & 2x_1/x_2^3 \end{pmatrix}.$$ 

This (symmetric) matrix is called the **Hessian**.
The function

\[ f(x) + \nabla f(x)^T h + \frac{1}{2} h^T \nabla^2 f(x) h \]

is quadratic in \( h \) and approximates \( f(x + h) \) near the point \( h = 0 \) even better (second-order approximation).

\( f \) is convex iff \( \nabla^2 f \) is positive semidefinite, that is the last term above is always nonnegative.

**Example**

If \( n = 1 \) (one dimension) then \( f(x) \) is convex iff \( f''(x) \geq 0 \) and equivalently if \( f'(x) \) is nondecreasing.
Given convex functions $f_0, \ldots, f_m$, and vectors $a_i$, numbers $b_i$.

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \ i = 1, \ldots, m. \\
& \quad a_i^T x = b_i, \ i = 1, \ldots, k.
\end{align*}
\]

We will generally assume $k = 0$, for simplicity.

**Example**

Least squares, with extra per-variable bounds:

\[
\begin{align*}
\text{minimize} & \quad |Ax - b|^2 \\
\text{subject to} & \quad u_i \leq x_i \leq v_i, \ i = 1, \ldots, m.
\end{align*}
\]

Programs of this type are called **quadratic**.
A **generalized** convex program: minimize $f_0(x)$ provided $x \in C$ for some given convex body $C$.

**Example**

minimize $\sum_{ij} c_{ij} x_{ij}$

subject to $\sum_{ij} a_{ijk} x_{ij} = b_k$, $k = 1, \ldots, m$,

$X = (x_{ij})$ is positive semidefinite.

Programs of this type are called **semidefinite**.

Where do such problems arise?
Example

Given an \( n \times n \) matrix \( W = (w_{ij}) \) with \( w_{ij} \geq 0 \), choose unit vectors \( u_1, \ldots, u_n \) in \( \mathbb{R}^n \) such that \( \sum_{ij} w_{ij} u_i^T u_j \) is minimal:

\[
\text{minimize} \quad \sum_{ij} w_{ij} u_i^T u_j \\
\text{subject to} \quad u_i^T u_i = 1, \ i = 1, \ldots, n.
\]

The term \( u_i^T u_j \) measures is the cosine of the angle between \( u_i \) and \( u_j \).

We are trying to put the unit vectors \( u_i \) as apart from each other in terms of angle as possible. Apartness of certain pairs is more important than of others, as shown by the weights \( w_{ij} \) in \( w_{ij} u_i^T u_j \).
Writing $x_{ij} = u_i^T u_j$ the problem can be written as follows.

$$\text{minimize} \quad \sum_{ij} w_{ij} x_{ij}$$
$$\text{subject to} \quad x_{ii} = 1, \quad i = 1, \ldots, n,$$
$$X = (x_{ij}) \quad \text{is positive semidefinite.}$$

Indeed, each feasible solution of the previous problem gives rise to a matrix a $U$ with columns $u_i$. The matrix $X = U^T U$ is a feasible solution to the new problem.

Conversely, any positive semidefinite matrix $X$ can be written as $U^T U$. Denoting the columns of $U$ as $u_i$, also $1 = x_{ii} = u_i^T u$, so the $u_i$ are unit vectors.

We will see an application of this program later to get a famous approximate solution to the maximum cut problem.
If $f(x)$ is a differentiable convex function in a convex domain $C$ and the minimum $x^*$ is in the interior then

$$\nabla f(x^*) = 0$$

is a necessary and sufficient condition for the minimum. So we can view the optimization problem as the problem of solving this equation.
Suppose that, for a monotonic function \( f : [a, b] \rightarrow \mathbb{R} \) with \( f(a) < 0 \), \( f(b) > 0 \) we want to find \( x \) with \( f(x) = 0 \). How to do this if we can only ask for values of \( f(x) \) at each step? Let \( a_0 = a \), \( b_0 = b \). We find sequences \( a_0 < a_1 < \cdots \) and \( b_0 > b_1 > \cdots \) with \( f(a_k) \leq 0 \leq f(b_k) \). We stop if \( f(b_k) - f(a_k) < \varepsilon \):

**Definition (Binary search)**

Rule: let \( x^{(k)} = (a_k + b_k)/2 \). If \( f(x^{(k)}) \leq 0 \) let \( a_{k+1} = x^{(k)} \), \( b_{k+1} = b_k \), else \( a_{k+1} = a_k \), \( b_{k+1} = x^{(k)} \).

With this method \( b_k - a_k = (b - a)/2^k \), so each step adds a bit to the exactness (number of significant bits) of the solution.
If the derivatives of $f(x)$ are also available and $f$ behaves nicely, then we can do much better.

**Definition (Newton’s method)**

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}.$$  

This is much faster than binary search in case it works. To understand it, recall $f(x^{(k)} + h) \approx f(x^{(k)}) + hf'(x^{(k)})$. Then $x^{(k+1)} = x^{(k)} + h$ is the point at which the approximation $f(x^{(k)}) + hf'(x^{(k)})$ hits 0.

It can be shown that under certain conditions, (when $f'(x)$ does not change too much) each step of Newton’s method doubles the number of significant bits of the solution. We call this quadratic convergence.
Unconstrained minimization by descent methods

Given a convex function \( f(x) \) over \( \mathbb{R}^n \), and an initial point \( x^0 \), a group of optimization methods (descent methods) creates a sequence of points \( x^{(1)}, x^{(2)}, \ldots \), such that \( f(x^{(k+1)}) < f(x^{(k)}) \). We stop when \( f(x^{(k)}) - f(x^{(k+1)}) \) becomes sufficiently small. More precisely, we set

\[
x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)}.
\]

Here, \( \Delta x^{(k)} \) is the kth search step (a vector), which we multiply by the step length \( t^{(k)} \).

Example (Gradient search)

Let the search step be \( \Delta x^{(k)} = -\nabla f(x^{(k)}) \).
This seems reasonable (for sufficiently small step length \( t^{(k)} \)), since \( f(x) \) decreases fastest in the direction of the vector \( -\nabla f(x) \).
But gradient search can be very slow.

**Example**

Let $f(x,y) = 10^{-5}x^2 + y^2$, then the minimum is at $x = y = 0$. Let us start with $x^{(0)} = 10^2$, $y^{(0)} = 1$. The gradient at point $(x,y)^T$ is $(2 \cdot 10^{-5}x, 2y)^T$.

So, at $(x_0,y_0)^T$ it is $(2 \cdot 10^{-3}, 2)^T$ (almost vertical), even though direction to the the minimum (in $(0,0)$) is almost horizontal. If we choose $(x^{(1)},y^{(1)})^T = (x^{(0)},y^{(0)})^T - t\nabla f(x^{(0)},y^{(0)})$ with $t = 1$ then $x^{(1)} = 10^2 - 2 \cdot 10^{-3}$, $y^{(1)} = -1$. Continuing this way would

1. have us jump forever between $y = -1$ and $y = 1$, and
2. decrease $x$ too slowly.

To avoid problem 1, the step length $t$ must be chosen carefully. To avoid problem 2, something other than the gradient method is needed.
Let \( x = x^{(k)}, t = t^{(k)}, x' = x^{(k+1)} = x + t\Delta x \).

How to choose the step length \( t \)? We want \( f(x) - f(x') \) to be a good fraction of \( t\nabla f(x)^T \Delta x \). We know this is true if \( t \) is small enough, so we just keep decreasing \( t \) until we have this:

**Definition (Backtracking line search)**

Choose \( \alpha = 0.2, \beta = 0.5 \) and initially, \( t = 1 \).
While \( f(x) - f(x') < \alpha t\nabla f(x)^T \Delta x \), set \( t := \beta t \).
Better search direction: Newton’s method.
One-dimensional Newton method for minimization:

\[ x^{(k+1)} = x^{(k)} - f'(x^{(k)}) / f''(x^{(k)}) , \]

since we are seeking the root of \( f'(x) \).
The \( n \)-dimensional analog gives the search direction

\[ \Delta x = -\nabla^2 f(x)^{-1} \nabla f(x) . \]

Recall the second-order approximation

\[ f(x) + \nabla f(x)^T h + \frac{1}{2} h^T \nabla^2 f(x) h . \]

Its minimum, found by differentiation, is \( x - \Delta x \).
Indeed, let $\nabla^2 f(x) = A = (a_{ij})$, $\nabla f(x) = b = (b_i)$, then we need to minimize

$$
\sum_i b_i h_i + \frac{1}{2} \sum_{ij} a_{ij} h_i h_j = \sum_i b_i h_i + \frac{1}{2} \sum_i a_{ii} h_i^2 + \sum_i \sum_{j>i} a_{ij} h_i h_j.
$$

Differenciating by $h_k$ gives

$$
b_k + a_{kk} h_k + \sum_{j \neq k} a_{kj} h_j = b_k + \sum_j a_{kj} h_j.
$$

This is 0 for all $k$ if

$$
b + Ah = 0,
$$

$$
h = -A^{-1} b.
$$
In summary, a reasonable unconstrained optimization method for twice differenciable convex functions is using Newton’s method, along with backtracking line search. See estimates of convergence in the book (Boyd-Vanderberghe). Essentially, each Newton step before the start of quadratic convergence makes a constant-size progress towards the optimum. The size of the constant depends on some overall bounds of the sort

\[ mI \leq \nabla^2 f(x) \leq MI \]

(where \( A \preceq B \) means that \( B - A \) is positive semidefinite) or on some special properties of the function \( f \) (self-concordance).
Recall the convex optimization problem with constraints:

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m.
\end{align*}
\]

Let

\[ p^* \]

\[ f_0(x) + \sum_{i} \lambda_i f_i(x) \]

for \( \lambda \geq 0 \).
The variables $\lambda_i$ are called dual variables: they put a “price” on each constraint, in the same way as $y_i$ did in the duality theory of linear programming. The dual function is defined by

$$g(\lambda) = \inf_x L(x, \lambda).$$

It is a concave function of $\lambda$ (infimum of linear functions). The dual optimization problem is

$$\text{maximize} \quad g(\lambda)$$
$$\text{subject to} \quad \lambda \geq 0.$$  

Again, each feasible solution of this problem lowerbounds $p^*$. Duality gap: the difference between the minimum of the primal and the maximum of the dual. Under certain conditions (which we assume satisfied here) this is 0.
Assume differentiability, and assume that $x^*$ and $\lambda^*$ are optimal for the primal and for the dual, with zero duality gap. Then

\[ \lambda^* \geq 0, \]
\[ f_i(x^*) \leq 0, \quad i = 1, \ldots, m, \]
\[ \lambda^*_i f_i(x^*) = 0, \quad i = 1, \ldots, m \]

(complementary slackness),

\[ \nabla_x L(x^*, \lambda^*) = 0 \]

that is

\[ \nabla f_0(x) + \sum_i \lambda_i^* \nabla f(x^*) = 0. \]

These conditions are also sufficient for optimum and for zero duality gap. They are called the KKT conditions (Karush-Kuhn-Tucker).
Adapt unconstrained minimization to constraints. Introduce penalty for approaching the walls:

$$\phi(x) = -\sum_{i=1}^{m} \ln(-f_i(x)).$$

Fix a large parameter $t$ and minimize

$$f_0(x) + (1/t)\phi(x).$$

Suppose that we start from a feasible point $x^{(0)}$ and use some descent method. As $x^{(k)}$ is approaching a “wall”, some $f_i(x^{(k)})$ is approaching 0, so $\phi(x)$ is approaching $\infty$. This keeps the descent inside the domain.

Such methods are called **interior-point methods**.
As we choose $t$ larger, the effect of the penalty becomes smaller (except very near the wall), so the optimum we find becomes closer to the optimum of the original constrained problem. It moves towards the true optimum along a central path.

**Example (Linear programming)**

\[
\begin{align*}
\text{minimize} \quad & c^T x \\
\text{subject to} \quad & a_i^T x \leq b_i, \quad i = 1, \ldots, m.
\end{align*}
\]

New form: minimize

\[
c^T x - \frac{1}{t} \sum_{i=1}^{m} \ln(b_i - a_i^T x). \]

Péter Gács (Boston University)
Duality gap along the central path

Note $\nabla \phi(x) = \sum_i (-1/f_i(x)) \nabla f_i(x)$. Let $x_t$ be a point along the central path. Then by the optimality condition,

$$\nabla f_0(x_t) + (1/t) \sum_i \frac{-1}{f_i(x_t)} \nabla f_i(x_t) = 0,$$

$$\nabla f_0(x_t) + \sum_i \lambda_i(t) \nabla f_i(x_t) = 0,$$

where $\lambda_i(t) = -1/(tf_i(x_t))$. This gives a feasible solution $\lambda(t)$ of the dual of the original problem, with dual objective value:

$$f_0(x_t) + \sum_{i=1}^m \lambda_i(t)f_i(x_t) = f_0(x_t) - m/t,$$

so the duality gap is $m/t$ (in the sense of the difference between some primal and some dual value).
Definition (Barrier method)

Assume that strictly feasible \( x \) is given, with \( t := t^{(0)} > 0 \) factor \( \mu = 10 \), tolerance \( \varepsilon > 0 \).

repeat

1. Centering. Compute \( x^*(t) \) by minimizing \( tf_0 + \phi \), starting at \( x \).
2. Update. \( x = x^*(t) \).
3. Stopping criterion. Quit if \( m/t < \varepsilon \).
4. Increase. \( t := \mu t \).

For the centering step, use for example Newton’s method. Why not a very large centering step? If \( \mu \) is too large then you get close to the boundary too soon, possibly at a wrong place, and each centering step will cost too many Newton steps. What starting value \( t^{(0)} \)? The initial duality gap \( m/t^{(0)} \) should be comparable to the expected total decrease \( f(x^{(0)}) - p^* \).
Under certain conditions ("self-concordance"), a constant upper bound on the number of Newton steps for each centering step.

**Linear programming**
The "self-concordance" conditions are satisfied for linear programming. So, in linear programming, $k$ centering steps require $O(k)$ Newton steps, which bring us $ck$ significant digits of the optimum, for $c > 0$.

Suppose we want to decide if the optimum is 0 (feasibility problem). We have seen that if the original linear programming problem had $m \cdot n$ coefficients of at most $r$ digits each, then a denominator in the optimum value has at most $mnr$ digits. It follows that if the optimum is known to precision more than $mnr$ digits, we will be able to decide whether it is 0.
Examples

- Shortest vs. longest simple paths
- Euler tour vs. Hamiltonian cycle
- 2-SAT vs. 3-SAT. Satisfiability for circuits and for conjunctive normal form (SAT). Reducing satisfiability for circuits to 3-SAT. Use of reduction in this course: proving hardness.
- Ultrasound test of sex of fetus.
Decision problems vs. optimization problems vs. search problems.

Example

Given a graph $G$.

**Decision**  Given $k$, does $G$ have an independent subset of size $\geq k$?

**Optimization**  What is the size of the largest independent set?

**Search**  Given $k$, give an independent set of size $k$ (if there is one).

**Optimization+search**  Give a maximum size independent set.
Random access machine

**Memory**: one-way infinite tape: cell \(i\) contains natural number \(T[i]\) of arbitrary size.

**Program**: a sequence of instructions, in the “program store”: a (potentially) infinite sequence of labeled registers containing instructions. A program counter.

Instruction types:
- \(T[T[i]] = T[T[j]]\) random access
- \(T[i] = T[j] \pm T[k]\) addition
- if \(R_0 > 0\) then jump to \(s\) conditional branching

The cost of an operation will be taken to be proportional to the total length of the numbers participating in it. This keeps the cost realistic despite the arbitrary size of numbers in the registers.
Abstract problems
Instance. Solution.

Encodings
Concrete problems: encoded into strings.
Polynomial-time computable functions, polynomial-time decidable sets.
Polynomially related encodings.
Language: a set of strings. Deciding a language.
An *NP problem* is defined with the help of a polynomial-time function

\[ V(x, w) \]

with yes/no values that verifies, for a given input \( x \) and witness (certificate) \( w \) whether \( w \) is indeed witness for \( x \).
The same decision problem may belong to very different verification functions (search problems).

**Example (Compositeness)**

Let the decision problem be the question whether a number $x$ is composite (nonprime). The obvious verifiable property is

$$V_1(x, w) \iff (1 < w < x) \land (w|x).$$

There is also a very different verifiable property $V_2(x, w)$ for compositeness such that, for a certain polynomial-time computable $b(x)$, if $x$ is composite then at least half of the numbers $1 \leq w \leq b(x)$ are witnesses. This can be used for probabilistic prime number tests.
Reduction of problem $A_1$ to problem $A_2$ in terms of the verification functions $V_1$, $V_2$ and a reduction (translation) function $\tau$:

$$\exists w V_1(x, w) \iff \exists u V_2(\tau(x), u).$$

Example
Reducing linear programming to solving a set of linear inequalities.

NP-hardness.
NP-completeness.
Theorem

Satisfiability is NP-complete.

Proof via circuit satisfiability.

Theorem

INDEPENDENT SET is NP-complete.

Reducing SAT to it.

Example

Integer linear programming. In particular, the subset sum problem.

Reduction of 3SAT to subset sum.

Example

Set cover $\geq$ vertex cover $\sim$ independent set.
In case of NP-complete problems, maybe something can be said about how well we can approximate a solution. We will formulate the question only for problems, where we maximize a positive function. For object function $f(x,y)$ for $x,y \in \{0, 1\}^n$, the optimum is

$$M(x) = \max_y f(x,y)$$

where $y$ runs over the possible “witnesses”.

For $0 < \lambda$, an algorithm $A(x)$ is a $\lambda$-approximation if

$$f(x, A(x)) > M(x)/\lambda.$$ 

For minimization problems, with minimum $m(x)$, we require

$$f(x, A(x)) < M(x)\lambda.$$
Try local improvements as long as you can.

**Example (Maximum cut)**

Repeat: find a point on one side of the cut whose moving to the other side increases the cutsizes.

**Theorem**

*If you cannot improve anymore with this algorithm then you are within a factor 2 of the optimum.*

**Proof.** The unimprovable cut contains at least half of all edges.

**Question** Suppose that edges have weights. The greedy algorithm still brings within factor 2 of the optimum. But does it take a polynomial number of steps?
We mentioned that semidefinite programming gives a better approximation. We actually did not show how to do semidefinite programming, since we did not introduce a penalty function for the requirement that a matrix $X$ is positive semidefinite. (It turns out that a good penalty function is $-\log \det X$.)
Less greed is sometimes better

What does the greedy algorithm for vertex cover say? The following, less greedy algorithm has better performance guarantee.

\[
\text{Approx\_Vertex\_Cover}(G) \\
C \leftarrow \emptyset \\
E' \leftarrow E[G] \\
\text{while } E' \neq \emptyset \text{ do} \\
\quad \text{let } (u, v) \text{ be an arbitrary edge in } E' \\
\quad C \leftarrow C \cup \{u, v\} \\
\quad \text{remove from } E' \text{ every edge incident on either } u \text{ or } v \\
\text{return } C
\]
Theorem

Approx_Vertex_Cover has a ratio bound of 2.

Proof. The points of C are endpoints of a matching. Any optimum vertex cover must contain half of them.
More general vertex cover problem for $G = (V, E)$, with weigh $w_i$ in vertex $i$. Let $x_i = 1$ if vertex $x$ is selected. Linear programming problem without the integrality condition:

$$\begin{align*}
\text{minimize} & \quad w^T x \\
\text{subject to} & \quad x_i + x_j \geq 1, (i, j) \in E, \\
& \quad x \geq 0.
\end{align*}$$

Let the optimal solution be $x^*$. Choose $\bar{x}_i = 1$ if $x_i^* \geq 1/2$ and 0 otherwise.

**Claim**

*Solution $\bar{x}$ has approximation ratio 2.*

**Proof.** We increased each $x_i^*$ by at most a factor of 2.
The set-covering problem

Given $(X, \mathcal{F})$: a set $X$ and a family $\mathcal{F}$ of subsets of $X$, find a min-size subset of $\mathcal{F}$ covering $X$.

Example: Smallest committee with people covering all skills.

Generalization: Set $S$ has weight $w_S > 0$. We want a minimum-weight set cover.

**Greedy_Set_Cover**$(X, \mathcal{F})$

\[
\begin{align*}
U &\leftarrow X \\
\mathcal{C} &\leftarrow \emptyset \\
\text{while } U \neq \emptyset \text{ do} \\
&\quad \text{select an } S \in \mathcal{F} \text{ that maximizes } |S \cap U|/w_S \\
&\quad U \leftarrow U \setminus S \\
&\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{S\} \\
\text{return } \mathcal{C}
\end{align*}
\]
Approximations
The set-covering problem

Analysis

Let \( H(n) = 1 + 1/2 + \cdots + 1/n (\approx \ln n) \).

**Theorem**

*Greedy_Set_Cover* has a ratio bound \( \max_{S \in \mathcal{F}} H(|S|) \).

Let \( S_i \) be the \( i \)-th set selected by the algorithm. For a set \( S \) let

\[
u_i(S) = |S \setminus (S_1 \cup \cdots \cup S_i)|.\]

If element \( x \) is about to be covered then its share of cost of covering it by the set \( S \), distributed among all new elements covered by \( S \), is \( w_S / u_{i-1}(S) \). The greedy algorithm chooses the set for which this cost is smallest:

\[
c_x = w_{S_i} / u_{i-1}(S_i) = \min_{S \in \mathcal{F}} w_S / u_{i-1}(S).\]

Total cost: \( \sum_x c_x = \sum_{S \in \mathcal{C}} w_S \).
Lemma

For all $S$ in $\mathcal{F}$ we have $\sum_{x \in S} c_x \leq w_S H(|S|)$.

Proof. Let $k$ be first $i$ with $u_i(S) = 0$.

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} \sum_{x \in S_i \cap S} c_x = \sum_{i=1}^{k} (u_{i-1}(S) - u_i(S)) \frac{w_{S_i}}{u_{i-1}(S_i)}.$$ 

Since $w_{S_i}/u_{i-1}(S_i) \leq w_S/u_{i-1}(S)$ (greed), this is

$$\leq \sum_{i=1}^{k} (u_{i-1}(S) - u_i(S)) \frac{w_S}{u_{i-1}(S)} = w_S \sum_{i=1}^{k} \frac{u_{i-1}(S) - u_i(S)}{u_{i-1}(S)}.$$ 

For all integers $a > b > 0$ we have

$$\frac{b-a}{b} = \sum_{i=a+1}^{b} \frac{1}{i} \leq \sum_{i=a+1}^{b} \frac{1}{i} = H(b) - H(a)$$

hence the last term above is

$$\leq w_S \sum_{i=1}^{u_0(S)} 1/i = w_S H(|S|).$$
Proof of the theorem. Let $\mathcal{C}_*$ be the optimal set cover and $\mathcal{C}$ the cover returned by the algorithm.

$$\sum_{x} c_x \leq \sum_{S \in \mathcal{C}_*} \sum_{x \in S} c_x \leq \sum_{S \in \mathcal{C}_*} w_S H(|S|) \leq H(|S^*|) \sum_{S \in \mathcal{C}_*} w_S$$

where $S^*$ is the largest set.
A linear programming interpretation

Let for set $S$ be $p_S = 1$ if $S$ is selected and 0 otherwise. Set cover problem, without integrality condition:

\[
\begin{align*}
\text{minimize} & \quad \sum_S w_S p_S \\
\text{subject to} & \quad \sum_{S \ni x} p_S \geq 1, \quad x \in X, \\
& \quad p_S \geq 0, \quad S \in \mathcal{F},
\end{align*}
\]

Dual with variables $c_x, x \in X$:

\[
\begin{align*}
\text{maximize} & \quad \sum_{x \in X} c_x \\
\text{subject to} & \quad \sum_{x \in S} c_x \leq w_S, \quad S \in \mathcal{F}, \\
& \quad c_x \geq 0, \quad x \in X.
\end{align*}
\]

The greedy choice made $c_x$ possibly larger than allowed here. The lemma bounds the factor by which it is larger, which is the same factor by which the greedy choice can be worse than the optimum.
An algorithm that for every $\epsilon$, gives an $(1 + \epsilon)$-approximation.

- A problem is **fully approximable** if it has a polynomial-time approximation scheme.  
  Example: see a version KNAPSACK below.

- It is **partly approximable** if there is a lower bound $\lambda_{\text{min}} > 1$ on the achievable approximation ratio.  
  Example: MAXIMUM CUT, VERTEX COVER, MAX-SAT.

- It is **inapproximable** if even this cannot be achieved.  
  Example: INDEPENDENT SET (deep result). The approximation status of this problem is different from VERTEX COVER, despite the close equivalence between the two problems.
Fully approximable version of knapsack

Given: \( b > a_1 \geq a_2 \geq \ldots \geq a_n \).

maximize \( \mathbf{a}^T \mathbf{x} \)
subject to \( \mathbf{a}^T \mathbf{x} \leq b \),
\[ x_i = 0, 1, \ i = 1, \ldots, n. \]

Idea for approximation: break each \( a_i \) and \( b \) into a smaller number of big chunks, and use dynamic programming. Let \( r > 0 \), \( a''_i = \lfloor a_i / r \rfloor \).

maximize \( (a'')^T \mathbf{x} \)
subject to \( \mathbf{a}^T \mathbf{x} \leq b \),
\[ x_i = 0, 1, \ i = 1, \ldots, n. \]
For the optimal solution $x''$ of the changed problem, estimate $a^T x^* / a^T x''$. We have

$$a^T x'' / r \geq (a'')^T x'' \geq (a'')^T x^* \geq (a/r)^T x^* - n,$$

$$a^T x'' \geq a^T x^* - rn.$$

Let $r = \epsilon a_1 / n$, then

$$(a'')^T x'' \geq a^T x^* - \epsilon a_1,$$

$$\frac{(a'')^T x''}{a^T x^*} \geq 1 - \frac{\epsilon a_1}{a^T x^*} \geq 1 - \epsilon.$$

The amount of time is of the order of

$$nb / r = n^2 b / (a_1 \epsilon) \leq n^3 / \epsilon,$$

which is polynomial for each fixed $\epsilon$. 