Appendices

Appendix A

Mathematical background

In this appendix we give a brief review of some basic concepts from analysis and linear algebra. The treatment is by no means complete, and is meant mostly to set out our notation.

A.1 Norms

A.1.1 Inner product, Euclidean norm, and angle

The *standard inner product* on $\mathbf{R}^n$, the set of real n-vectors, is given by

$$\langle x, y \rangle = x^T y = \sum_{i=1}^{n} x_i y_i,$$

for $x, y \in \mathbf{R}^n$. In this book we use the notation $x^T y$, instead of $\langle x, y \rangle$. The *Euclidean norm*, or ℓ_2-norm, of a vector $x \in \mathbf{R}^n$ is defined as

$$\|x\|_2 = (x^T x)^{1/2} = (x_1^2 + \cdots + x_n^2)^{1/2}. \tag{A.1}$$

The *Cauchy-Schwartz inequality* states that $|x^T y| \leq \|x\|_2 \|y\|_2$ for any $x, y \in \mathbf{R}^n$. The (unsigned) *angle* between nonzero vectors $x, y \in \mathbf{R}^n$ is defined as

$$\angle(x, y) = \cos^{-1}\left(\frac{x^T y}{\|x\|_2 \|y\|_2}\right),$$

where we take $\cos^{-1}(u) \in [0, \pi]$. We say x and y are *orthogonal* if $x^T y = 0$.

The standard inner product on $\mathbf{R}^{m \times n}$, the set of $m \times n$ real matrices, is given by

$$\langle X, Y \rangle = \mathbf{tr}(X^T Y) = \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} Y_{ij},$$

for $X, Y \in \mathbf{R}^{m \times n}$. (Here $\mathbf{tr}$ denotes *trace* of a matrix, *i.e.*, the sum of its diagonal elements.) We use the notation $\mathbf{tr}(X^T Y)$ instead of $\langle X, Y \rangle$. Note that the inner

product of two matrices is the inner product of the associated vectors, in $\mathbf{R}^{mn}$, obtained by listing the coefficients of the matrices in some order, such as row major.

The *Frobenius norm* of a matrix $X \in \mathbf{R}^{m\times n}$ is given by

$$\|X\|_F = \left(\mathbf{tr}(X^T X)\right)^{1/2} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}^2\right)^{1/2}. \tag{A.2}$$

The Frobenius norm is the Euclidean norm of the vector obtained by listing the coefficients of the matrix. (The ℓ_2-norm of a matrix is a different norm; see §A.1.5.)

The standard inner product on $\mathbf{S}^n$, the set of symmetric $n \times n$ matrices, is given by

$$\langle X, Y\rangle = \mathbf{tr}(XY) = \sum_{i=1}^{n}\sum_{j=1}^{n} X_{ij}Y_{ij} = \sum_{i=1}^{n} X_{ii}Y_{ii} + 2\sum_{i<j} X_{ij}Y_{ij}.$$

A.1.2 Norms, distance, and unit ball

A function $f : \mathbf{R}^n \to \mathbf{R}$ with $\mathbf{dom}\, f = \mathbf{R}^n$ is called a *norm* if

- f is nonnegative: $f(x) \geq 0$ for all $x \in \mathbf{R}^n$
- f is definite: $f(x) = 0$ only if $x = 0$
- f is homogeneous: $f(tx) = |t|f(x)$, for all $x \in \mathbf{R}^n$ and $t \in \mathbf{R}$
- f satisfies the triangle inequality: $f(x + y) \leq f(x) + f(y)$, for all $x, y \in \mathbf{R}^n$

We use the notation $f(x) = \|x\|$, which is meant to suggest that a norm is a generalization of the absolute value on $\mathbf{R}$. When we specify a particular norm, we use the notation $\|x\|_{\mathrm{symb}}$, where the subscript is a mnemonic to indicate which norm is meant.

A norm is a measure of the *length* of a vector x; we can measure the *distance* between two vectors x and y as the length of their difference, *i.e.*,

$$\mathbf{dist}(x, y) = \|x - y\|.$$

We refer to $\mathbf{dist}(x, y)$ as the distance between x and y, in the norm $\|\cdot\|$.

The set of all vectors with norm less than or equal to one,

$$\mathcal{B} = \{x \in \mathbf{R}^n \mid \|x\| \leq 1\},$$

is called the *unit ball* of the norm $\|\cdot\|$. The unit ball satisfies the following properties:

- $\mathcal{B}$ is symmetric about the origin, *i.e.*, $x \in \mathcal{B}$ if and only if $-x \in \mathcal{B}$
- $\mathcal{B}$ is convex
- $\mathcal{B}$ is closed, bounded, and has nonempty interior

Conversely, if $C \subseteq \mathbf{R}^n$ is any set satisfying these three conditions, then it is the unit ball of a norm, which is given by

$$\|x\| = (\sup\{t \geq 0 \mid tx \in C\})^{-1}.$$

A.1.3 Examples

The simplest example of a norm is the absolute value on $\mathbf{R}$. Another simple example is the Euclidean or ℓ_2-norm on $\mathbf{R}^n$, defined above in (A.1). Two other frequently used norms on $\mathbf{R}^n$ are the *sum-absolute-value*, or ℓ_1*-norm*, given by

$$\|x\|_1 = |x_1| + \cdots + |x_n|,$$

and the *Chebyshev* or ℓ_∞*-norm*, given by

$$\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}.$$

These three norms are part of a family parametrized by a constant traditionally denoted p, with $p \geq 1$: the ℓ_p*-norm* is defined by

$$\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}.$$

This yields the ℓ_1-norm when $p = 1$ and the Euclidean norm when $p = 2$. It is easy to show that for any $x \in \mathbf{R}^n$,

$$\lim_{p\to\infty} \|x\|_p = \max\{|x_1|, \ldots, |x_n|\},$$

so the ℓ_∞-norm also fits in this family, as a limit.

Another important family of norms are the *quadratic norms*. For $P \in \mathbf{S}^n_{++}$, we define the P-quadratic norm as

$$\|x\|_P = (x^T P x)^{1/2} = \|P^{1/2}x\|_2.$$

The unit ball of a quadratic norm is an ellipsoid (and conversely, if the unit ball of a norm is an ellipsoid, the norm is a quadratic norm).

Some common norms on $\mathbf{R}^{m\times n}$ are the Frobenius norm, defined above in (A.2), the sum-absolute-value norm,

$$\|X\|_{\text{sav}} = \sum_{i=1}^{m}\sum_{j=1}^{n} |X_{ij}|,$$

and the maximum-absolute-value norm,

$$\|X\|_{\text{mav}} = \max\{|X_{ij}| \mid i = 1, \ldots, m,\ j = 1, \ldots, n\}.$$

We will encounter several other important norms of matrices in §A.1.5.

A.1.4 Equivalence of norms

Suppose that $\|\cdot\|_{\mathrm{a}}$ and $\|\cdot\|_{\mathrm{b}}$ are norms on $\mathbf{R}^n$. A basic result of analysis is that there exist positive constants α and β such that, for all $x \in \mathbf{R}^n$,

$$\alpha\|x\|_{\mathrm{a}} \leq \|x\|_{\mathrm{b}} \leq \beta\|x\|_{\mathrm{a}}.$$

This means that the norms are *equivalent*, *i.e.*, they define the same set of open subsets, the same set of convergent sequences, and so on (see §A.2). (We conclude that any norms on any finite-dimensional vector space are equivalent, but on infinite-dimensional vector spaces, the result need not hold.) Using convex analysis, we can give a more specific result: If $\|\cdot\|$ is any norm on $\mathbf{R}^n$, then there exists a quadratic norm $\|\cdot\|_P$ for which

$$\|x\|_P \leq \|x\| \leq \sqrt{n}\|x\|_P$$

holds for all x. In other words, any norm on $\mathbf{R}^n$ can be uniformly approximated, within a factor of $\sqrt{n}$, by a quadratic norm. (See §8.4.1.)

A.1.5 Operator norms

Suppose $\|\cdot\|_{\mathrm{a}}$ and $\|\cdot\|_{\mathrm{b}}$ are norms on $\mathbf{R}^m$ and $\mathbf{R}^n$, respectively. We define the *operator norm* of $X \in \mathbf{R}^{m\times n}$, induced by the norms $\|\cdot\|_{\mathrm{a}}$ and $\|\cdot\|_{\mathrm{b}}$, as

$$\|X\|_{\mathrm{a,b}} = \sup\{\|Xu\|_{\mathrm{a}} \mid \|u\|_{\mathrm{b}} \leq 1\}.$$

(It can be shown that this defines a norm on $\mathbf{R}^{m\times n}$.)

When $\|\cdot\|_{\mathrm{a}}$ and $\|\cdot\|_{\mathrm{b}}$ are both Euclidean norms, the operator norm of X is its *maximum singular value*, and is denoted $\|X\|_2$:

$$\|X\|_2 = \sigma_{\max}(X) = (\lambda_{\max}(X^TX))^{1/2}.$$

(This agrees with the Euclidean norm on $\mathbf{R}^m$, when $X \in \mathbf{R}^{m\times 1}$, so there is no clash of notation.) This norm is also called the *spectral norm* or *ℓ_2-norm* of X.

As another example, the norm induced by the ℓ_∞-norm on $\mathbf{R}^m$ and $\mathbf{R}^n$, denoted $\|X\|_\infty$, is the *max-row-sum norm*,

$$\|X\|_\infty = \sup\{\|Xu\|_\infty \mid \|u\|_\infty \leq 1\} = \max_{i=1,\ldots,m} \sum_{j=1}^{n} |X_{ij}|.$$

The norm induced by the ℓ_1-norm on $\mathbf{R}^m$ and $\mathbf{R}^n$, denoted $\|X\|_1$, is the *max-column-sum norm*,

$$\|X\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{m} |X_{ij}|.$$

A.1.6 Dual norm

Let $\|\cdot\|$ be a norm on $\mathbf{R}^n$. The associated *dual norm*, denoted $\|\cdot\|_*$, is defined as

$$\|z\|_* = \sup\{z^T x \mid \|x\| \leq 1\}.$$

(This can be shown to be a norm.) The dual norm can be interpreted as the operator norm of z^T, interpreted as a $1 \times n$ matrix, with the norm $\|\cdot\|$ on $\mathbf{R}^n$, and the absolute value on $\mathbf{R}$:

$$\|z\|_* = \sup\{|z^T x| \mid \|x\| \leq 1\}.$$

From the definition of dual norm we have the inequality

$$z^T x \leq \|x\| \, \|z\|_*,$$

which holds for all x and z. This inequality is tight, in the following sense: for any x there is a z for which the inequality holds with equality. (Similarly, for any z there is an x that gives equality.) The dual of the dual norm is the original norm: we have $\|x\|_{**} = \|x\|$ for all x. (This need not hold in infinite-dimensional vector spaces.)

The dual of the Euclidean norm is the Euclidean norm, since

$$\sup\{z^T x \mid \|x\|_2 \leq 1\} = \|z\|_2.$$

(This follows from the Cauchy-Schwarz inequality; for nonzero z, the value of x that maximizes $z^T x$ over $\|x\|_2 \leq 1$ is $z/\|z\|_2$.)

The dual of the ℓ_1-norm is the ℓ_∞-norm:

$$\sup\{z^T x \mid \|x\|_\infty \leq 1\} = \sum_{i=1}^{n} |z_i| = \|z\|_1,$$

and the dual of the ℓ_∞-norm is the ℓ_1-norm. More generally, the dual of the ℓ_p-norm is the ℓ_q-norm, where q satisfies $1/p + 1/q = 1$, *i.e.*, $q = p/(p-1)$.

As another example, consider the ℓ_2- or spectral norm on $\mathbf{R}^{m \times n}$. The associated dual norm is

$$\|Z\|_{2*} = \sup\{\mathbf{tr}(Z^T X) \mid \|X\|_2 \leq 1\},$$

which turns out to be the sum of the singular values,

$$\|Z\|_{2*} = \sigma_1(Z) + \cdots + \sigma_r(Z) = \mathbf{tr}(Z^T Z)^{1/2},$$

where $r = \mathbf{rank}\, Z$. This norm is sometimes called the *nuclear* norm.

A.2 Analysis

A.2.1 Open and closed sets

An element $x \in C \subseteq \mathbf{R}^n$ is called an *interior* point of C if there exists an $\epsilon > 0$ for which

$$\{y \mid \|y - x\|_2 \leq \epsilon\} \subseteq C,$$

i.e., there exists a ball centered at x that lies entirely in C. The set of all points interior to C is called the *interior* of C and is denoted $\mathbf{int}\, C$. (Since all norms on $\mathbf{R}^n$ are equivalent to the Euclidean norm, all norms generate the same set of interior points.) A set C is *open* if $\mathbf{int}\, C = C$, *i.e.*, every point in C is an interior point. A set $C \subseteq \mathbf{R}^n$ is *closed* if its complement $\mathbf{R}^n \setminus C = \{x \in \mathbf{R}^n \mid x \notin C\}$ is open.

The *closure* of a set C is defined as

$$\mathbf{cl}\, C = \mathbf{R}^n \setminus \mathbf{int}(\mathbf{R}^n \setminus C),$$

i.e., the complement of the interior of the complement of C. A point x is in the closure of C if for every $\epsilon > 0$, there is a $y \in C$ with $\|x - y\|_2 \leq \epsilon$.

We can also describe closed sets and the closure in terms of convergent sequences and limit points. A set C is closed if and only if it contains the limit point of every convergent sequence in it. In other words, if $x_1, x_2, \ldots$ converges to x, and $x_i \in C$, then $x \in C$. The closure of C is the set of all limit points of convergent sequences in C.

The *boundary* of the set C is defined as

$$\mathbf{bd}\, C = \mathbf{cl}\, C \setminus \mathbf{int}\, C.$$

A *boundary point* x (*i.e.*, a point $x \in \mathbf{bd}\, C$) satisfies the following property: For all $\epsilon > 0$, there exists $y \in C$ and $z \notin C$ with

$$\|y - x\|_2 \leq \epsilon, \qquad \|z - x\|_2 \leq \epsilon,$$

i.e., there exist arbitrarily close points in C, and also arbitrarily close points not in C. We can characterize closed and open sets in terms of the boundary operation: C is *closed* if it contains its boundary, *i.e.*, $\mathbf{bd}\, C \subseteq C$. It is *open* if it contains no boundary points, *i.e.*, $C \cap \mathbf{bd}\, C = \emptyset$.

A.2.2 Supremum and infimum

Suppose $C \subseteq \mathbf{R}$. A number a is an *upper bound* on C if for each $x \in C$, $x \leq a$. The set of upper bounds on a set C is either empty (in which case we say C is unbounded above), all of $\mathbf{R}$ (only when $C = \emptyset$), or a closed infinite interval $[b, \infty)$. The number b is called the *least upper bound* or *supremum* of the set C, and is denoted $\sup C$. We take $\sup \emptyset = -\infty$, and $\sup C = \infty$ if C is unbounded above. When $\sup C \in C$, we say the supremum of C is attained or achieved.

When the set C is finite, $\sup C$ is the maximum of its elements. Some authors use the notation $\max C$ to denote supremum, when it is attained, but we follow standard mathematical convention, using $\max C$ only when the set C is finite.

We define lower bound, and infimum, in a similar way. A number a is a lower bound on $C \subseteq \mathbf{R}$ if for each $x \in C$, $a \leq x$. The *infimum* (or *greatest lower bound*) of a set $C \subseteq \mathbf{R}$ is defined as $\inf C = -\sup(-C)$. When C is finite, the infimum is the minimum of its elements. We take $\inf \emptyset = \infty$, and $\inf C = -\infty$ if C is unbounded below, *i.e.*, has no lower bound.

A.3 Functions

A.3.1 Function notation

Our notation for functions is mostly standard, with one exception. When we write

$$f : A \to B$$

we mean that f is a function on the set $\mathbf{dom}\, f \subseteq A$ into the set B; in particular we can have $\mathbf{dom}\, f$ a proper subset of the set A. Thus the notation $f : \mathbf{R}^n \to \mathbf{R}^m$ means that f maps (some) n-vectors into m-vectors; it does not mean that $f(x)$ is defined for every $x \in \mathbf{R}^n$. This convention is similar to function declarations in computer languages. Specifying the data types of the input and output arguments of a function gives the *syntax* of that function; it does not guarantee that any input argument with the specified data type is valid.

As an example consider the function $f : \mathbf{S}^n \to \mathbf{R}$, given by

$$f(X) = \log \det X, \tag{A.3}$$

with $\mathbf{dom}\, f = \mathbf{S}^n_{++}$. The notation $f : \mathbf{S}^n \to \mathbf{R}$ specifies the *syntax* of f: it takes as argument a symmetric $n \times n$ matrix, and returns a real number. The notation $\mathbf{dom}\, f = \mathbf{S}^n_{++}$ specifies which symmetric $n \times n$ matrices are valid input arguments for f (*i.e.*, only positive definite ones). The formula (A.3) specifies what $f(X)$ is, for $X \in \mathbf{dom}\, f$.

A.3.2 Continuity

A function $f : \mathbf{R}^n \to \mathbf{R}^m$ is *continuous* at $x \in \mathbf{dom}\, f$ if for all $\epsilon > 0$ there exists a δ such that

$$y \in \mathbf{dom}\, f, \quad \|y - x\|_2 \leq \delta \implies \|f(y) - f(x)\|_2 \leq \epsilon.$$

Continuity can be described in terms of limits: whenever the sequence $x_1, x_2, \ldots$ in $\mathbf{dom}\, f$ converges to a point $x \in \mathbf{dom}\, f$, the sequence $f(x_1), f(x_2), \ldots$ converges to $f(x)$, *i.e.*,

$$\lim_{i\to\infty} f(x_i) = f(\lim_{i\to\infty} x_i).$$

A function f is *continuous* if it is continuous at every point in its domain.

A.3.3 Closed functions

A function $f : \mathbf{R}^n \to \mathbf{R}$ is said to be *closed* if, for each $\alpha \in \mathbf{R}$, the sublevel set

$$\{x \in \mathbf{dom}\, f \mid f(x) \leq \alpha\}$$

is closed. This is equivalent to the condition that the epigraph of f,

$$\mathbf{epi}\, f = \{(x, t) \in \mathbf{R}^{n+1} \mid x \in \mathbf{dom}\, f,\ f(x) \leq t\},$$

is closed. (This definition is general, but is usually only applied to convex functions.)

If $f : \mathbf{R}^n \to \mathbf{R}$ is continuous, and $\mathbf{dom}\, f$ is closed, then f is closed. If $f : \mathbf{R}^n \to \mathbf{R}$ is continuous, with $\mathbf{dom}\, f$ open, then f is closed if and only if f converges to ∞ along every sequence converging to a boundary point of $\mathbf{dom}\, f$. In other words, if $\lim_{i\to\infty} x_i = x \in \mathbf{bd}\,\mathbf{dom}\, f$, with $x_i \in \mathbf{dom}\, f$, we have $\lim_{i\to\infty} f(x_i) = \infty$.

Example A.1 *Examples on* $\mathbf{R}$.

- The function $f : \mathbf{R} \to \mathbf{R}$, with $f(x) = x \log x$, $\mathbf{dom}\, f = \mathbf{R}_{++}$, is *not* closed.
- The function $f : \mathbf{R} \to \mathbf{R}$, with

$$f(x) = \begin{cases} x \log x & x > 0 \\ 0 & x = 0, \end{cases} \qquad \mathbf{dom}\, f = \mathbf{R}_+,$$

  is closed.
- The function $f(x) = -\log x$, $\mathbf{dom}\, f = \mathbf{R}_{++}$, is closed.

A.4 Derivatives

A.4.1 Derivative and gradient

Suppose $f : \mathbf{R}^n \to \mathbf{R}^m$ and $x \in \mathbf{int}\,\mathbf{dom}\, f$. The *derivative* (or *Jacobian*) of f at x is the matrix $Df(x) \in \mathbf{R}^{m\times n}$, given by

$$Df(x)_{ij} = \frac{\partial f_i(x)}{\partial x_j}, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n,$$

provided the partial derivatives exist. If the partial derivatives exist, we say f is *differentiable* at x. The function f is *differentiable* if $\mathbf{dom}\, f$ is open, and it is differentiable at every point in its domain.

The affine function of z given by

$$f(x) + Df(x)(z - x)$$

is called the *first-order approximation* of f at (or near) x. Evidently this function agrees with f at $z = x$; when z is *close* to x, this affine function is *very close* to f:

$$\lim_{z\in\mathbf{dom}\, f,\ z\neq x,\ z\to x} \frac{\|f(z) - f(x) - Df(x)(z - x)\|_2}{\|z - x\|_2} = 0. \tag{A.4}$$

The derivative matrix $Df(x)$ is the *only* matrix in $\mathbf{R}^{m\times n}$ that satisfies the condition (A.4). This gives an alternative method for finding the derivative, by deriving a first-order approximation of the function f at x.

Gradient

When f is real-valued (*i.e.*, $f : \mathbf{R}^n \to \mathbf{R}$) the derivative $Df(x)$ is a $1 \times n$ matrix, *i.e.*, it is a *row* vector. Its transpose is called the *gradient* of the function:

$$\nabla f(x) = Df(x)^T,$$

which is a (column) vector, *i.e.*, in $\mathbf{R}^n$. Its components are the partial derivatives of f:

$$\nabla f(x)_i = \frac{\partial f(x)}{\partial x_i}, \quad i = 1, \ldots, n.$$

The first-order approximation of f at a point $x \in \mathbf{int}\,\mathbf{dom}\, f$ can be expressed as (the affine function of z)

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

Examples

As a simple example consider the quadratic function $f : \mathbf{R}^n \to \mathbf{R}$,

$$f(x) = (1/2)x^T P x + q^T x + r,$$

where $P \in \mathbf{S}^n$, $q \in \mathbf{R}^n$, and $r \in \mathbf{R}$. Its derivative at x is the row vector $Df(x) = x^T P + q^T$, and its gradient is

$$\nabla f(x) = Px + q.$$

As a more interesting example, we consider the function $f : \mathbf{S}^n \to \mathbf{R}$, given by

$$f(X) = \log\det X, \qquad \mathbf{dom}\, f = \mathbf{S}^n_{++}.$$

One (tedious) way to find the gradient of f is to introduce a basis for $\mathbf{S}^n$, find the gradient of the associated function, and finally translate the result back to $\mathbf{S}^n$. Instead, we will directly find the first-order approximation of f at $X \in \mathbf{S}^n_{++}$. Let $Z \in \mathbf{S}^n_{++}$ be close to X, and let $\Delta X = Z - X$ (which is assumed to be small). We have

$$\begin{aligned}
\log\det Z &= \log\det(X + \Delta X) \\
&= \log\det\left(X^{1/2}(I + X^{-1/2}\Delta X X^{-1/2})X^{1/2}\right) \\
&= \log\det X + \log\det(I + X^{-1/2}\Delta X X^{-1/2}) \\
&= \log\det X + \sum_{i=1}^n \log(1 + \lambda_i),
\end{aligned}$$

where λ_i is the ith eigenvalue of $X^{-1/2}\Delta X X^{-1/2}$. Now we use the fact that ΔX is small, which implies λ_i are small, so to first order we have $\log(1 + \lambda_i) \approx \lambda_i$. Using this first-order approximation in the expression above, we get

$$\begin{aligned}
\log\det Z &\approx \log\det X + \sum_{i=1}^n \lambda_i \\
&= \log\det X + \mathbf{tr}(X^{-1/2}\Delta X X^{-1/2}) \\
&= \log\det X + \mathbf{tr}(X^{-1}\Delta X) \\
&= \log\det X + \mathbf{tr}\left(X^{-1}(Z - X)\right),
\end{aligned}$$

where we have used the fact that the sum of the eigenvalues is the trace, and the property $\mathbf{tr}(AB) = \mathbf{tr}(BA)$.

Thus, the first-order approximation of f at X is the affine function of Z given by

$$f(Z) \approx f(X) + \mathbf{tr}\left(X^{-1}(Z - X)\right).$$

Noting that the second term on the righthand side is the standard inner product of X^{-1} and $Z - X$, we can identify X^{-1} as the gradient of f at X. Thus, we can write the simple formula

$$\nabla f(X) = X^{-1}.$$

This result should not be surprising, since the derivative of $\log x$, on $\mathbf{R}_{++}$, is $1/x$.

A.4.2 Chain rule

Suppose $f : \mathbf{R}^n \to \mathbf{R}^m$ is differentiable at $x \in \mathbf{int}\,\mathbf{dom}\, f$ and $g : \mathbf{R}^m \to \mathbf{R}^p$ is differentiable at $f(x) \in \mathbf{int}\,\mathbf{dom}\, g$. Define the composition $h : \mathbf{R}^n \to \mathbf{R}^p$ by $h(z) = g(f(z))$. Then h is differentiable at x, with derivative

$$Dh(x) = Dg(f(x))Df(x). \tag{A.5}$$

As an example, suppose $f : \mathbf{R}^n \to \mathbf{R}$, $g : \mathbf{R} \to \mathbf{R}$, and $h(x) = g(f(x))$. Taking the transpose of $Dh(x) = Dg(f(x))Df(x)$ yields

$$\nabla h(x) = g'(f(x))\nabla f(x). \tag{A.6}$$

Composition with affine function

Suppose $f : \mathbf{R}^n \to \mathbf{R}^m$ is differentiable, $A \in \mathbf{R}^{n\times p}$, and $b \in \mathbf{R}^n$. Define $g : \mathbf{R}^p \to \mathbf{R}^m$ as $g(x) = f(Ax + b)$, with $\mathbf{dom}\, g = \{x \mid Ax + b \in \mathbf{dom}\, f\}$. The derivative of g is, by the chain rule (A.5), $Dg(x) = Df(Ax + b)A$.

When f is real-valued (*i.e.*, $m = 1$), we obtain the formula for the gradient of a composition of a function with an affine function,

$$\nabla g(x) = A^T \nabla f(Ax + b).$$

For example, suppose that $f : \mathbf{R}^n \to \mathbf{R}$, $x, v \in \mathbf{R}^n$, and we define the function $\tilde{f} : \mathbf{R} \to \mathbf{R}$ by $\tilde{f}(t) = f(x + tv)$. (Roughly speaking, $\tilde{f}$ is f, restricted to the line $\{x + tv \mid t \in \mathbf{R}\}$.) Then we have

$$D\tilde{f}(t) = \tilde{f}'(t) = \nabla f(x + tv)^T v.$$

(The scalar $\tilde{f}'(0)$ is the *directional derivative* of f, at x, in the direction v.)

Example A.2 Consider the function $f : \mathbf{R}^n \to \mathbf{R}$, with $\mathbf{dom}\, f = \mathbf{R}^n$ and

$$f(x) = \log \sum_{i=1}^{m} \exp(a_i^T x + b_i),$$

where $a_1, \ldots, a_m \in \mathbf{R}^n$, and $b_1, \ldots, b_m \in \mathbf{R}$. We can find a simple expression for its gradient by noting that it is the composition of the affine function $Ax + b$, where $A \in \mathbf{R}^{m \times n}$ with rows $a_1^T, \ldots, a_m^T$, and the function $g : \mathbf{R}^m \to \mathbf{R}$ given by $g(y) = \log(\sum_{i=1}^m \exp y_i)$. Simple differentiation (or the formula (A.6)) shows that

$$\nabla g(y) = \frac{1}{\sum_{i=1}^m \exp y_i} \begin{bmatrix} \exp y_1 \\ \vdots \\ \exp y_m \end{bmatrix}, \tag{A.7}$$

so by the composition formula we have

$$\nabla f(x) = \frac{1}{\mathbf{1}^T z} A^T z$$

where $z_i = \exp(a_i^T x + b_i)$, $i = 1, \ldots, m$.

Example A.3 We derive an expression for $\nabla f(x)$, where

$$f(x) = \log\det(F_0 + x_1 F_1 + \cdots + x_n F_n),$$

where $F_0, \ldots, F_n \in \mathbf{S}^p$, and

$$\mathbf{dom}\, f = \{x \in \mathbf{R}^n \mid F_0 + x_1 F_1 + \cdots + x_n F_n \succ 0\}.$$

The function f is the composition of the affine mapping from $x \in \mathbf{R}^n$ to $F_0 + x_1 F_1 + \cdots + x_n F_n \in \mathbf{S}^p$, with the function $\log\det X$. We use the chain rule to evaluate

$$\frac{\partial f(x)}{\partial x_i} = \mathbf{tr}(F_i \nabla \log\det(F)) = \mathbf{tr}(F^{-1} F_i),$$

where $F = F_0 + x_1 F_1 + \cdots + x_n F_n$. Thus we have

$$\nabla f(x) = \begin{bmatrix} \mathbf{tr}(F^{-1} F_1) \\ \vdots \\ \mathbf{tr}(F^{-1} F_n) \end{bmatrix}.$$

A.4.3 Second derivative

In this section we review the second derivative of a real-valued function $f : \mathbf{R}^n \to \mathbf{R}$. The second derivative or *Hessian matrix* of f at $x \in \mathbf{int}\,\mathbf{dom}\, f$, denoted $\nabla^2 f(x)$, is given by

$$\nabla^2 f(x)_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}, \qquad i = 1, \ldots n, \quad j = 1, \ldots, n,$$

provided f is twice differentiable at x, where the partial derivatives are evaluated at x. The *second-order approximation* of f, at or near x, is the quadratic function of z defined by

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

This second-order approximation satisfies

$$\lim_{z \in \mathbf{dom}\, f,\ z \neq x,\ z \to x} \frac{|f(z) - \widehat{f}(z)|}{\|z - x\|_2^2} = 0.$$

Not surprisingly, the second derivative can be interpreted as the derivative of the first derivative. If f is differentiable, the *gradient mapping* is the function $\nabla f : \mathbf{R}^n \to \mathbf{R}^n$, with $\mathbf{dom}\, \nabla f = \mathbf{dom}\, f$, with value $\nabla f(x)$ at x. The derivative of this mapping is

$$D\nabla f(x) = \nabla^2 f(x).$$

Examples

As a simple example consider the quadratic function $f : \mathbf{R}^n \to \mathbf{R}$,

$$f(x) = (1/2)x^T P x + q^T x + r,$$

where $P \in \mathbf{S}^n$, $q \in \mathbf{R}^n$, and $r \in \mathbf{R}$. Its gradient is $\nabla f(x) = Px + q$, so its Hessian is given by $\nabla^2 f(x) = P$. The second-order approximation of a quadratic function is itself.

As a more complicated example, we consider again the function $f : \mathbf{S}^n \to \mathbf{R}$, given by $f(X) = \log\det X$, with $\mathbf{dom}\, f = \mathbf{S}^n_{++}$. To find the second-order approximation (and therefore, the Hessian), we will derive a first-order approximation of the gradient, $\nabla f(X) = X^{-1}$. For $Z \in \mathbf{S}^n_{++}$ near $X \in \mathbf{S}^n_{++}$, and $\Delta X = Z - X$, we have

$$\begin{aligned}
Z^{-1} &= (X + \Delta X)^{-1} \\
&= \left(X^{1/2}(I + X^{-1/2}\Delta X X^{-1/2})X^{1/2}\right)^{-1} \\
&= X^{-1/2}(I + X^{-1/2}\Delta X X^{-1/2})^{-1}X^{-1/2} \\
&\approx X^{-1/2}(I - X^{-1/2}\Delta X X^{-1/2})X^{-1/2} \\
&= X^{-1} - X^{-1}\Delta X X^{-1},
\end{aligned}$$

using the first-order approximation $(I + A)^{-1} \approx I - A$, valid for A small.

This approximation is enough for us to identify the Hessian of f at X. The Hessian is a quadratic form on $\mathbf{S}^n$. Such a quadratic form is cumbersome to describe in the general case, since it requires four indices. But from the first-order approximation of the gradient above, the quadratic form can be expressed as

$$-\mathbf{tr}(X^{-1}UX^{-1}V),$$

where $U, V \in \mathbf{S}^n$ are the arguments of the quadratic form. (This generalizes the expression for the scalar case: $(\log x)'' = -1/x^2$.)

Now we have the second-order approximation of f near X:

$$\begin{aligned}
f(Z) &= f(X + \Delta X) \\
&\approx f(X) + \mathbf{tr}(X^{-1}\Delta X) - (1/2)\,\mathbf{tr}(X^{-1}\Delta X X^{-1}\Delta X) \\
&\approx f(X) + \mathbf{tr}\left(X^{-1}(Z - X)\right) - (1/2)\,\mathbf{tr}\left(X^{-1}(Z - X)X^{-1}(Z - X)\right).
\end{aligned}$$

A.4.4 Chain rule for second derivative

A general chain rule for the second derivative is cumbersome in most cases, so we will state it only for some special cases that we will need.

Composition with scalar function

Suppose $f : \mathbf{R}^n \to \mathbf{R}$, $g : \mathbf{R} \to \mathbf{R}$, and $h(x) = g(f(x))$. Simply working out the partial derivatives yields

$$\nabla^2 h(x) = g'(f(x))\nabla^2 f(x) + g''(f(x))\nabla f(x)\nabla f(x)^T. \tag{A.8}$$

Composition with affine function

Suppose $f : \mathbf{R}^n \to \mathbf{R}$, $A \in \mathbf{R}^{n\times m}$, and $b \in \mathbf{R}^n$. Define $g : \mathbf{R}^m \to \mathbf{R}$ by $g(x) = f(Ax + b)$. Then we have

$$\nabla^2 g(x) = A^T \nabla^2 f(Ax + b)A.$$

As an example, consider the restriction of a real-valued function f to a line, *i.e.*, the function $\tilde{f}(t) = f(x + tv)$, where x and v are fixed. Then we have

$$\nabla^2 \tilde{f}(t) = \tilde{f}''(t) = v^T \nabla^2 f(x + tv)v.$$

Example A.4 We consider the function $f : \mathbf{R}^n \to \mathbf{R}$ from example A.2,

$$f(x) = \log \sum_{i=1}^{m} \exp(a_i^T x + b_i),$$

where $a_1, \ldots, a_m \in \mathbf{R}^n$, and $b_1, \ldots, b_m \in \mathbf{R}$. By noting that $f(x) = g(Ax+b)$, where $g(y) = \log(\sum_{i=1}^m \exp y_i)$, we can obtain a simple formula for the Hessian of f. Taking partial derivatives, or using the formula (A.8), noting that g is the composition of log with $\sum_{i=1}^m \exp y_i$, yields

$$\nabla^2 g(y) = \mathbf{diag}(\nabla g(y)) - \nabla g(y)\nabla g(y)^T,$$

where $\nabla g(y)$ is given in (A.7). By the composition formula we have

$$\nabla^2 f(x) = A^T \left(\frac{1}{\mathbf{1}^T z} \mathbf{diag}(z) - \frac{1}{(\mathbf{1}^T z)^2} z z^T \right) A,$$

where $z_i = \exp(a_i^T x + b_i)$, $i = 1, \ldots, m$.

A.5 Linear algebra

A.5.1 Range and nullspace

Let $A \in \mathbf{R}^{m\times n}$ (*i.e.*, A is a real matrix with m rows and n columns). The *range* of A, denoted $\mathcal{R}(A)$, is the set of all vectors in $\mathbf{R}^m$ that can be written as linear

combinations of the columns of A, *i.e.*,

$$\mathcal{R}(A) = \{Ax \mid x \in \mathbf{R}^n\}.$$

The range $\mathcal{R}(A)$ is a subspace of $\mathbf{R}^m$, *i.e.*, it is itself a vector space. Its dimension is the *rank* of A, denoted $\mathbf{rank}\, A$. The rank of A can never be greater than the minimum of m and n. We say A has *full rank* if $\mathbf{rank}\, A = \min\{m, n\}$.

The *nullspace* (or *kernel*) of A, denoted $\mathcal{N}(A)$, is the set of all vectors x mapped into zero by A:

$$\mathcal{N}(A) = \{x \mid Ax = 0\}.$$

The nullspace is a subspace of $\mathbf{R}^n$.

Orthogonal decomposition induced by A

If $\mathcal{V}$ is a subspace of $\mathbf{R}^n$, its *orthogonal complement*, denoted $\mathcal{V}^\perp$, is defined as

$$\mathcal{V}^\perp = \{x \mid z^T x = 0 \text{ for all } z \in \mathcal{V}\}.$$

(As one would expect of a complement, we have $\mathcal{V}^{\perp\perp} = \mathcal{V}$.)

A basic result of linear algebra is that, for any $A \in \mathbf{R}^{m\times n}$, we have

$$\mathcal{N}(A) = \mathcal{R}(A^T)^\perp.$$

(Applying the result to A^T we also have $\mathcal{R}(A) = \mathcal{N}(A^T)^\perp$.) This result is often stated as

$$\mathcal{N}(A) \overset{\perp}{\oplus} \mathcal{R}(A^T) = \mathbf{R}^n. \tag{A.9}$$

Here the symbol $\overset{\perp}{\oplus}$ refers to *orthogonal direct sum*, *i.e.*, the sum of two subspaces that are orthogonal. The decomposition (A.9) of $\mathbf{R}^n$ is called the *orthogonal decomposition induced by* A.

A.5.2 Symmetric eigenvalue decomposition

Suppose $A \in \mathbf{S}^n$, *i.e.*, A is a real symmetric $n \times n$ matrix. Then A can be factored as

$$A = Q\Lambda Q^T, \tag{A.10}$$

where $Q \in \mathbf{R}^{n\times n}$ is *orthogonal*, *i.e.*, satisfies $Q^T Q = I$, and $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n)$. The (real) numbers λ_i are the *eigenvalues* of A, and are the roots of the *characteristic polynomial* $\det(sI - A)$. The columns of Q form an orthonormal set of *eigenvectors* of A. The factorization (A.10) is called the *spectral decomposition* or (symmetric) *eigenvalue decomposition* of A.

We order the eigenvalues as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. We use the notation $\lambda_i(A)$ to refer to the ith largest eigenvalue of $A \in \mathbf{S}$. We usually write the largest or maximum eigenvalue as $\lambda_1(A) = \lambda_{\max}(A)$, and the least or minimum eigenvalue as $\lambda_n(A) = \lambda_{\min}(A)$.

The determinant and trace can be expressed in terms of the eigenvalues,

$$\det A = \prod_{i=1}^{n} \lambda_i, \qquad \mathbf{tr}\, A = \sum_{i=1}^{n} \lambda_i,$$

as can the spectral and Frobenius norms,

$$\|A\|_2 = \max_{i=1,\ldots,n} |\lambda_i| = \max\{\lambda_1, -\lambda_n\}, \qquad \|A\|_F = \left(\sum_{i=1}^{n} \lambda_i^2\right)^{1/2}.$$

Definiteness and matrix inequalities

The largest and smallest eigenvalues satisfy

$$\lambda_{\max}(A) = \sup_{x \neq 0} \frac{x^T A x}{x^T x}, \qquad \lambda_{\min}(A) = \inf_{x \neq 0} \frac{x^T A x}{x^T x}.$$

In particular, for any x, we have

$$\lambda_{\min}(A) x^T x \leq x^T A x \leq \lambda_{\max}(A) x^T x,$$

with both inequalities tight for (different) choices of x.

A matrix $A \in \mathbf{S}^n$ is called *positive definite* if for all $x \neq 0$, $x^T A x > 0$. We denote this as $A \succ 0$. By the inequality above, we see that $A \succ 0$ if and only all its eigenvalues are positive, *i.e.*, $\lambda_{\min}(A) > 0$. If $-A$ is positive definite, we say A is *negative definite*, which we write as $A \prec 0$. We use $\mathbf{S}^n_{++}$ to denote the set of positive definite matrices in $\mathbf{S}^n$.

If A satisfies $x^T A x \geq 0$ for all x, we say that A is *positive semidefinite* or *nonnegative definite*. If $-A$ is in nonnegative definite, *i.e.*, if $x^T A x \leq 0$ for all x, we say that A is *negative semidefinite* or *nonpositive definite*. We use $\mathbf{S}^n_+$ to denote the set of nonnegative definite matrices in $\mathbf{S}^n$.

For $A, B \in \mathbf{S}^n$, we use $A \prec B$ to mean $B - A \succ 0$, and so on. These inequalities are called *matrix inequalities*, or generalized inequalities associated with the positive semidefinite cone.

Symmetric squareroot

Let $A \in \mathbf{S}^n_+$, with eigenvalue decomposition $A = Q\,\mathbf{diag}(\lambda_1, \ldots, \lambda_n)Q^T$. We define the (symmetric) squareroot of A as

$$A^{1/2} = Q\,\mathbf{diag}(\lambda_1^{1/2}, \ldots, \lambda_n^{1/2})Q^T.$$

The squareroot $A^{1/2}$ is the unique symmetric positive semidefinite solution of the equation $X^2 = A$.

A.5.3 Generalized eigenvalue decomposition

The *generalized eigenvalues* of a pair of symmetric matrices $(A, B) \in \mathbf{S}^n \times \mathbf{S}^n$ are defined as the roots of the polynomial $\det(sB - A)$.

We are usually interested in matrix pairs with $B \in \mathbf{S}^n_{++}$. In this case the generalized eigenvalues are also the eigenvalues of $B^{-1/2}AB^{-1/2}$ (which are real). As with the standard eigenvalue decomposition, we order the generalized eigenvalues in nonincreasing order, as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, and denote the maximum generalized eigenvalue by $\lambda_{\max}(A, B)$.

When $B \in \mathbf{S}^n_{++}$, the pair of matrices can be factored as

$$A = V\Lambda V^T, \qquad B = VV^T, \tag{A.11}$$

where $V \in \mathbf{R}^{n\times n}$ is nonsingular, and $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n)$, where λ_i are the generalized eigenvalues of the pair (A, B). The decomposition (A.11) is called the *generalized eigenvalue decomposition*.

The generalized eigenvalue decomposition is related to the standard eigenvalue decomposition of the matrix $B^{-1/2}AB^{-1/2}$. If $Q\Lambda Q^T$ is the eigenvalue decomposition of $B^{-1/2}AB^{-1/2}$, then (A.11) holds with $V = B^{1/2}Q$.

A.5.4 Singular value decomposition

Suppose $A \in \mathbf{R}^{m\times n}$ with $\mathbf{rank}\, A = r$. Then A can be factored as

$$A = U\Sigma V^T, \tag{A.12}$$

where $U \in \mathbf{R}^{m\times r}$ satisfies $U^TU = I$, $V \in \mathbf{R}^{n\times r}$ satisfies $V^TV = I$, and $\Sigma = \mathbf{diag}(\sigma_1, \ldots, \sigma_r)$, with

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0.$$

The factorization (A.12) is called the *singular value decomposition* (SVD) of A. The columns of U are called *left singular vectors* of A, the columns of V are *right singular vectors*, and the numbers σ_i are the *singular values*. The singular value decomposition can be written

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

where $u_i \in \mathbf{R}^m$ are the left singular vectors, and $v_i \in \mathbf{R}^n$ are the right singular vectors.

The singular value decomposition of a matrix A is closely related to the eigenvalue decomposition of the (symmetric, nonnegative definite) matrix A^TA. Using (A.12) we can write

$$A^TA = V\Sigma^2V^T = \begin{bmatrix} V & \tilde{V} \end{bmatrix} \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V & \tilde{V} \end{bmatrix}^T,$$

where $\tilde{V}$ is any matrix for which $[V\ \tilde{V}]$ is orthogonal. The righthand expression is the eigenvalue decomposition of A^TA, so we conclude that its nonzero eigenvalues are the singular values of A squared, and the associated eigenvectors of A^TA are the right singular vectors of A. A similar analysis of AA^T shows that its nonzero

eigenvalues are also the squares of the singular values of A, and the associated eigenvectors are the left singular vectors of A.

The first or largest singular value is also written as $\sigma_{\max}(A)$. It can be expressed as

$$\sigma_{\max}(A) = \sup_{x,y\neq 0} \frac{x^T A y}{\|x\|_2\|y\|_2} = \sup_{y\neq 0} \frac{\|Ay\|_2}{\|y\|_2}.$$

The righthand expression shows that the maximum singular value is the ℓ_2 operator norm of A. The *minimum singular value* of $A \in \mathbf{R}^{m\times n}$ is given by

$$\sigma_{\min}(A) = \begin{cases} \sigma_r(A) & r = \min\{m,n\} \\ 0 & r < \min\{m,n\}, \end{cases}$$

which is positive if and only if A is full rank.

The singular values of a symmetric matrix are the absolute values of its nonzero eigenvalues, sorted into descending order. The singular values of a symmetric positive semidefinite matrix are the same as its nonzero eigenvalues.

The *condition number* of a nonsingular $A \in \mathbf{R}^{n\times n}$, denoted $\mathbf{cond}(A)$ or $\kappa(A)$, is defined as

$$\mathbf{cond}(A) = \|A\|_2\|A^{-1}\|_2 = \sigma_{\max}(A)/\sigma_{\min}(A).$$

Pseudo-inverse

Let $A = U\Sigma V^T$ be the singular value decomposition of $A \in \mathbf{R}^{m\times n}$, with $\mathbf{rank}\, A = r$. We define the *pseudo-inverse* or *Moore-Penrose inverse* of A as

$$A^\dagger = V\Sigma^{-1}U^T \in \mathbf{R}^{n\times m}.$$

Alternative expressions are

$$A^\dagger = \lim_{\epsilon\to 0}(A^TA + \epsilon I)^{-1}A^T = \lim_{\epsilon\to 0} A^T(AA^T + \epsilon I)^{-1},$$

where the limits are taken with $\epsilon > 0$, which ensures that the inverses in the expressions exist. If $\mathbf{rank}\, A = n$, then $A^\dagger = (A^TA)^{-1}A^T$. If $\mathbf{rank}\, A = m$, then $A^\dagger = A^T(AA^T)^{-1}$. If A is square and nonsingular, then $A^\dagger = A^{-1}$.

The pseudo-inverse comes up in problems involving least-squares, minimum norm, quadratic minimization, and (Euclidean) projection. For example, $A^\dagger b$ is a solution of the least-squares problem

$$\text{minimize} \quad \|Ax - b\|_2^2$$

in general. When the solution is not unique, $A^\dagger b$ gives the solution with minimum (Euclidean) norm. As another example, the matrix $AA^\dagger = UU^T$ gives (Euclidean) projection on $\mathcal{R}(A)$. The matrix $A^\dagger A = VV^T$ gives (Euclidean) projection on $\mathcal{R}(A^T)$.

The optimal value $p^\star$ of the (general, nonconvex) quadratic optimization problem

$$\text{minimize} \quad (1/2)x^TPx + q^Tx + r,$$

where $P \in \mathbf{S}^n$, can be expressed as

$$p^\star = \begin{cases} -(1/2)q^TP^\dagger q + r & P \succeq 0, \quad q \in \mathcal{R}(P) \\ -\infty & \text{otherwise.} \end{cases}$$

(This generalizes the expression $p^\star = -(1/2)q^TP^{-1}q + r$, valid for $P \succ 0$.)

A.5.5 Schur complement

Consider a matrix $X \in \mathbf{S}^n$ partitioned as

$$X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix},$$

where $A \in \mathbf{S}^k$. If $\det A \neq 0$, the matrix

$$S = C - B^T A^{-1} B$$

is called the *Schur complement* of A in X. Schur complements arise in several contexts, and appear in many important formulas and theorems. For example, we have

$$\det X = \det A \det S.$$

Inverse of block matrix

The Schur complement comes up in solving linear equations, by eliminating one block of variables. We start with

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix},$$

and assume that $\det A \neq 0$. If we eliminate x from the top block equation and substitute it into the bottom block equation, we obtain $v = B^T A^{-1} u + Sy$, so

$$y = S^{-1}(v - B^T A^{-1} u).$$

Substituting this into the first equation yields

$$x = \left(A^{-1} + A^{-1} B S^{-1} B^T A^{-1}\right) u - A^{-1} B S^{-1} v.$$

We can express these two equations as a formula for the inverse of a block matrix:

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1} B S^{-1} B^T A^{-1} & -A^{-1} B S^{-1} \\ -S^{-1} B^T A^{-1} & S^{-1} \end{bmatrix}.$$

In particular, we see that the Schur complement is the inverse of the $2,2$ block entry of the inverse of X.

Minimization and definiteness

The Schur complement arises when you minimize a quadratic form over some of the variables. Suppose $A \succ 0$, and consider the minimization problem

$$\text{minimize} \quad u^T A u + 2 v^T B^T u + v^T C v \tag{A.13}$$

with variable u. The solution is $u = -A^{-1} B v$, and the optimal value is

$$\inf_u \begin{bmatrix} u \\ v \end{bmatrix}^T \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = v^T S v. \tag{A.14}$$

From this we can derive the following characterizations of positive definiteness or semidefiniteness of the block matrix X:

- $X \succ 0$ if and only if $A \succ 0$ and $S \succ 0$.
- If $A \succ 0$, then $X \succeq 0$ if and only if $S \succeq 0$.

Schur complement with singular A

Some Schur complement results have generalizations to the case when A is singular, although the details are more complicated. As an example, if $A \succeq 0$ and $Bv \in \mathcal{R}(A)$, then the quadratic minimization problem (A.13) (with variable u) is solvable, and has optimal value

$$v^T(C - B^T A^\dagger B)v,$$

where $A^\dagger$ is the pseudo-inverse of A. The problem is unbounded if $Bv \notin \mathcal{R}(A)$ or if $A \not\succeq 0$.

The range condition $Bv \in \mathcal{R}(A)$ can also be expressed as $(I - AA^\dagger)Bv = 0$, so we have the following characterization of positive semidefiniteness of the block matrix X:

$$X \succeq 0 \quad \Longleftrightarrow \quad A \succeq 0, \quad (I - AA^\dagger)B = 0, \quad C - B^T A^\dagger B \succeq 0.$$

Here the matrix $C - B^T A^\dagger B$ serves as a generalization of the Schur complement, when A is singular.

Bibliography

Some basic references for the material in this appendix are Rudin [Rud76] for analysis, and Strang [Str80] and Meyer [Mey00] for linear algebra. More advanced linear algebra texts include Horn and Johnson [HJ85, HJ91], Parlett [Par98], Golub and Van Loan [GL89], Trefethen and Bau [TB97], and Demmel [Dem97].

The concept of closed function (§A.3.3) appears frequently in convex optimization, although the terminology varies. The term is used by Rockafellar [Roc70, page 51], Hiriart-Urruty and Lemaréchal [HUL93, volume 1, page 149], Borwein and Lewis [BL00, page 76], and Bertsekas, Nedić, and Ozdaglar [Ber03, page 28].

Appendix B

Problems involving two quadratic functions

In this appendix we consider some optimization problems that involve two quadratic, but not necessarily convex, functions. Several strong results hold for these problems, even when they are not convex.

B.1 Single constraint quadratic optimization

We consider the problem with one constraint

$$\begin{array}{ll} \text{minimize} & x^T A_0 x + 2b_0^T x + c_0 \\ \text{subject to} & x^T A_1 x + 2b_1^T x + c_1 \leq 0, \end{array} \tag{B.1}$$

with variable $x \in \mathbf{R}^n$, and problem parameters $A_i \in \mathbf{S}^n$, $b_i \in \mathbf{R}^n$, $c_i \in \mathbf{R}$. We do not assume that $A_i \succeq 0$, so problem (B.1) is not a convex optimization problem.

The Lagrangian of (B.1) is

$$L(x, \lambda) = x^T(A_0 + \lambda A_1)x + 2(b_0 + \lambda b_1)^T x + c_0 + \lambda c_1,$$

and the dual function is

$$\begin{aligned} g(\lambda) &= \inf_x L(x, \lambda) \\ &= \begin{cases} c_0 + \lambda c_1 - (b_0 + \lambda b_1)^T (A_0 + \lambda A_1)^\dagger (b_0 + \lambda b_1) & A_0 + \lambda A_1 \succeq 0, \\ & \quad b_0 + \lambda b_1 \in \mathcal{R}(A_0 + \lambda A_1) \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

(see §A.5.4). Using a Schur complement, we can express the dual problem as

$$\begin{array}{ll} \text{maximize} & \gamma \\ \text{subject to} & \lambda \geq 0 \\ & \begin{bmatrix} A_0 + \lambda A_1 & b_0 + \lambda b_1 \\ (b_0 + \lambda b_1)^T & c_0 + \lambda c_1 - \gamma \end{bmatrix} \succeq 0, \end{array} \tag{B.2}$$

an SDP with two variables $\gamma, \lambda \in \mathbf{R}$.

The first result is that *strong duality holds* for problem (B.1) and its Lagrange dual (B.2), provided Slater's constraint qualification is satisfied, *i.e.*, there exists an x with $x^T A_1 x + 2b_1^T x + c_1 < 0$. In other words, if (B.1) is strictly feasible, the optimal values of (B.1) and (B.2) are equal. (A proof is given in §B.4.)

Relaxation interpretation

The dual of the SDP (B.2) is

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(A_0 X) + 2b_0^T x + c_0 \\ \text{subject to} & \mathbf{tr}(A_1 X) + 2b_1^T x + c_1 \leq 0 \\ & \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \succeq 0, \end{array} \tag{B.3}$$

an SDP with variables $X \in \mathbf{S}^n$, $x \in \mathbf{R}^n$. This dual SDP has an interesting interpretation in terms of the original problem (B.1).

We first note that (B.1) is equivalent to

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(A_0 X) + 2b_0^T x + c_0 \\ \text{subject to} & \mathbf{tr}(A_1 X) + 2b_1^T x + c_1 \leq 0 \\ & X = xx^T. \end{array} \tag{B.4}$$

In this formulation we express the quadratic terms $x^T A_i x$ as $\mathbf{tr}(A_i x x^T)$, and then introduce a new variable $X = xx^T$. Problem (B.4) has a linear objective function, one linear inequality constraint, and a nonlinear equality constraint $X = xx^T$. The next step is to replace the equality constraint by an inequality $X \succeq xx^T$:

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(A_0 X) + b_0^T x + c_0 \\ \text{subject to} & \mathbf{tr}(A_1 X) + b_1^T x + c_1 \leq 0 \\ & X \succeq xx^T. \end{array} \tag{B.5}$$

This problem is called a *relaxation* of (B.4), since we have replaced one of the constraints with a looser constraint. Finally we note that the inequality in (B.5) can be expressed as a linear matrix inequality by using a Schur complement, which gives (B.3).

A number of interesting facts follow immediately from this interpretation of (B.3) as a relaxation of (B.1). First, it is obvious that the optimal value of (B.3) is less than or equal to the optimal value of (B.1), since we minimize the same objective function over a larger set. Second, we can conclude that if $X = xx^T$ at the optimum of (B.3), then x must be optimal in (B.1).

Combining the result above, that strong duality holds between (B.1) and (B.2) (if (B.1) is strictly feasible), with strong duality between the dual SDPs (B.2) and (B.3), we conclude that strong duality holds between the original, nonconvex quadratic problem (B.1), and the SDP relaxation (B.3), provided (B.1) is strictly feasible.

B.2 The S-procedure

The next result is a theorem of alternatives for a pair of (nonconvex) quadratic inequalities. Let $A_1, A_2 \in \mathbf{S}^n$, $b_1, b_2 \in \mathbf{R}^n$, $c_1, c_2 \in \mathbf{R}$, and suppose there exists an $\hat{x}$ with

$$\hat{x}^T A_2 \hat{x} + 2b_2^T \hat{x} + c_2 < 0.$$

Then there exists an $x \in \mathbf{R}^n$ satisfying

$$x^T A_1 x + 2b_1^T x + c_1 < 0, \qquad x^T A_2 x + 2b_2^T x + c_2 \leq 0, \tag{B.6}$$

if and only if there exists no λ such that

$$\lambda \geq 0, \qquad \begin{bmatrix} A_1 & b_1 \\ b_1^T & c_1 \end{bmatrix} + \lambda \begin{bmatrix} A_2 & b_2 \\ b_2^T & c_2 \end{bmatrix} \succeq 0. \tag{B.7}$$

In other words, (B.6) and (B.7) are strong alternatives.

This result is readily shown to be equivalent to the result from §B.1, and a proof is given in §B.4. Here we point out that the two inequality systems are clearly weak alternatives, since (B.6) and (B.7) together lead to a contradiction:

$$\begin{aligned} 0 &\leq \begin{bmatrix} x \\ 1 \end{bmatrix}^T \left(\begin{bmatrix} A_1 & b_1 \\ b_1^T & c_1 \end{bmatrix} + \lambda \begin{bmatrix} A_2 & b_2 \\ b_2^T & c_2 \end{bmatrix} \right) \begin{bmatrix} x \\ 1 \end{bmatrix} \\ &= x^T A_1 x + 2b_1^T x + c_1 + \lambda(x^T A_2 x + 2b_2^T x + c_2) \\ &< 0. \end{aligned}$$

This theorem of alternatives is sometimes called the *S-procedure*, and is usually stated in the following form: the implication

$$x^T F_1 x + 2g_1^T x + h_1 \leq 0 \quad \Longrightarrow \quad x^T F_2 x + 2g_2^T x + h_2 \leq 0,$$

where $F_i \in \mathbf{S}^n$, $g_i \in \mathbf{R}^n$, $h_i \in \mathbf{R}$, holds if and only if there exists a λ such that

$$\lambda \geq 0, \qquad \begin{bmatrix} F_2 & g_2 \\ g_2^T & h_2 \end{bmatrix} \preceq \lambda \begin{bmatrix} F_1 & g_1 \\ g_1^T & h_1 \end{bmatrix},$$

provided there exists a point $\hat{x}$ with $\hat{x}^T F_1 \hat{x} + 2g_1^T \hat{x} + h_1 < 0$. (Note that sufficiency is clear.)

Example B.1 *Ellipsoid containment.* An ellipsoid $\mathcal{E} \subseteq \mathbf{R}^n$ with nonempty interior can be represented as the sublevel set of a quadratic function,

$$\mathcal{E} = \{x \mid x^T F x + 2g^T x + h \leq 0\},$$

where $F \in \mathbf{S}_{++}$ and $h - g^T F^{-1} g < 0$. Suppose $\tilde{\mathcal{E}}$ is another ellipsoid with similar representation,

$$\tilde{\mathcal{E}} = \{x \mid x^T \tilde{F} x + 2\tilde{g}^T x + \tilde{h} \leq 0\},$$

with $\tilde{F} \in \mathbf{S}_{++}$, $\tilde{h} - \tilde{g}^T \tilde{F}^{-1} \tilde{g} < 0$. By the S-procedure, we see that $\mathcal{E} \subseteq \tilde{\mathcal{E}}$ if and only if there is a $\lambda > 0$ such that

$$\begin{bmatrix} \tilde{F} & \tilde{g} \\ \tilde{g}^T & \tilde{h} \end{bmatrix} \preceq \lambda \begin{bmatrix} F & g \\ g^T & h \end{bmatrix}.$$

B.3 The field of values of two symmetric matrices

The following result is the basis for the proof of the strong duality result in §B.1 and the S-procedure in §B.2. If $A, B \in \mathbf{S}^n$, then for all $X \in \mathbf{S}^n_+$, there exists an $x \in \mathbf{R}^n$ such that

$$x^T A x = \mathbf{tr}(AX), \qquad x^T B x = \mathbf{tr}(BX). \tag{B.8}$$

Remark B.1 *Geometric interpretation.* This result has an interesting interpretation in terms of the set

$$W(A,B) = \{(x^T Ax, x^T Bx) \mid x \in \mathbf{R}^n\},$$

which is a cone in $\mathbf{R}^2$. It is the cone generated by the set

$$F(A,B) = \{(x^T Ax, x^T Bx) \mid \|x\|_2 = 1\},$$

which is called the *2-dimensional field of values* of the pair (A, B). Geometrically, $W(A, B)$ is the image of the set of rank-one positive semidefinite matrices under the linear transformation $f : \mathbf{S}^n \to \mathbf{R}^2$ defined by

$$f(X) = (\mathbf{tr}(AX), \mathbf{tr}(BX)).$$

The result that for every $X \in \mathbf{S}^n_+$ there exists an x satisfying (B.8) means that

$$W(A,B) = f(\mathbf{S}^n_+).$$

In other words, $W(A, B)$ is a *convex* cone.

The proof is constructive and uses induction on the rank of X. Suppose it is true for all $X \in \mathbf{S}^n_+$ with $1 \leq \mathbf{rank}\, X \leq k$, where $k \geq 2$, that there exists an x such that (B.8) holds. Then the result also holds if $\mathbf{rank}\, X = k+1$, as can be seen as follows. A matrix $X \in \mathbf{S}^n_+$ with $\mathbf{rank}\, X = k+1$ can be expressed as $X = yy^T + Z$ where $y \neq 0$ and $Z \in \mathbf{S}^n_+$ with $\mathbf{rank}\, Z = k$. By assumption, there exists a z such that $\mathbf{tr}(AZ) = z^T Az$, $\mathbf{tr}(AZ) = z^T Bz$. Therefore

$$\mathbf{tr}(AX) = \mathbf{tr}(A(yy^T + zz^T)), \qquad \mathbf{tr}(BX) = \mathbf{tr}(B(yy^T + zz^T)).$$

The rank of $yy^T + zz^T$ is one or two, so by assumption there exists an x such that (B.8) holds.

It is therefore sufficient to prove the result if $\mathbf{rank}\, X \leq 2$. If $\mathbf{rank}\, X = 0$ and $\mathbf{rank}\, X = 1$ there is nothing to prove. If $\mathbf{rank}\, X = 2$, we can factor X as $X = VV^T$ where $V \in \mathbf{R}^{n\times 2}$, with linearly independent columns v_1 and v_2. Without loss of generality we can assume that $V^T AV$ is diagonal. (If $V^T AV$ is not diagonal we replace V with VP where $V^T AV = P\,\mathbf{diag}(\lambda)P^T$ is the eigenvalue decomposition of $V^T AV$.) We will write $V^T AV$ and $V^T BV$ as

$$V^T AV = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \qquad V^T BV = \begin{bmatrix} \sigma_1 & \gamma \\ \gamma & \sigma_2 \end{bmatrix},$$

and define

$$w = \begin{bmatrix} \mathbf{tr}(AX) \\ \mathbf{tr}(BX) \end{bmatrix} = \begin{bmatrix} \lambda_1 + \lambda_2 \\ \sigma_1 + \sigma_2 \end{bmatrix}.$$

We need to show that $w = (x^T Ax, x^T Bx)$ for some x.

We distinguish two cases. First, assume $(0, \gamma)$ is a linear combination of the vectors (λ_1, σ_1) and (λ_2, σ_2):

$$0 = z_1\lambda_1 + z_2\lambda_2, \qquad \gamma = z_1\sigma_1 + z_2\sigma_2,$$

for some z_1, z_2. In this case we choose $x = \alpha v_1 + \beta v_2$, where α and β are determined by solving two quadratic equations in two variables

$$\alpha^2 + 2\alpha\beta z_1 = 1, \qquad \beta^2 + 2\alpha\beta z_2 = 1. \tag{B.9}$$

This will give the desired result, since

$$\begin{aligned}
&\begin{bmatrix} (\alpha v_1 + \beta v_2)^T A(\alpha v_1 + \beta v_2) \\ (\alpha v_1 + \beta v_2)^T B(\alpha v_1 + \beta v_2) \end{bmatrix} \\
&= \alpha^2 \begin{bmatrix} \lambda_1 \\ \sigma_1 \end{bmatrix} + 2\alpha\beta \begin{bmatrix} 0 \\ \gamma \end{bmatrix} + \beta^2 \begin{bmatrix} \lambda_2 \\ \sigma_2 \end{bmatrix} \\
&= (\alpha^2 + 2\alpha\beta z_1) \begin{bmatrix} \lambda_1 \\ \sigma_1 \end{bmatrix} + (\beta^2 + 2\alpha\beta z_2) \begin{bmatrix} \lambda_2 \\ \sigma_2 \end{bmatrix} \\
&= \begin{bmatrix} \lambda_1 + \lambda_2 \\ \sigma_1 + \sigma_2 \end{bmatrix}.
\end{aligned}$$

It remains to show that the equations (B.9) are solvable. To see this, we first note that α and β must be nonzero, so we can write the equations equivalently as

$$\alpha^2(1 + 2(\beta/\alpha)z_1) = 1, \qquad (\beta/\alpha)^2 + 2(\beta/\alpha)(z_2 - z_1) = 1.$$

The equation $t^2 + 2t(z_2 - z_1) = 1$ has a positive and a negative root. At least one of these roots (the root with the same sign as z_1) satisfies $1 + 2tz_1 > 0$, so we can choose

$$\alpha = \pm 1/\sqrt{1 + 2tz_1}, \qquad \beta = t\alpha.$$

This yields two solutions (α, β) that satisfy (B.9). (If both roots of $t^2 + 2t(z_2 - z_1) = 1$ satisfy $1 + 2tz_1 > 0$, we obtain four solutions.)

Next, assume that $(0, \gamma)$ is not a linear combination of (λ_1, σ_1) and (λ_2, σ_2). In particular, this means that (λ_1, σ_1) and (λ_2, σ_2) are linearly dependent. Therefore their sum $w = (\lambda_1 + \lambda_2, \sigma_1 + \sigma_2)$ is a nonnegative multiple of (λ_1, σ_1), or (λ_2, σ_2), or both. If $w = \alpha^2(\lambda_1, \sigma_1)$ for some α, we can choose $x = \alpha v_1$. If $w = \beta^2(\lambda_2, \sigma_2)$ for some β, we can choose $x = \beta v_2$.

B.4 Proofs of the strong duality results

We first prove the S-procedure result given in §B.2. The assumption of strict feasibility of $\hat{x}$ implies that the matrix

$$\begin{bmatrix} A_2 & b_2 \\ b_2^T & c_2 \end{bmatrix}$$

has at least one negative eigenvalue. Therefore

$$\tau \geq 0, \quad \tau \left[\begin{array}{cc} A_2 & b_2 \\ b_2^T & c_2 \end{array} \right] \succeq 0 \quad \Longrightarrow \quad \tau = 0.$$

We can apply the theorem of alternatives for nonstrict linear matrix inequalities, given in example 5.14, which states that (B.7) is infeasible if and only if

$$X \succeq 0, \qquad \mathbf{tr}\left(X \left[\begin{array}{cc} A_1 & b_1 \\ b_1^T & c_1 \end{array} \right] \right) < 0, \qquad \mathbf{tr}\left(X \left[\begin{array}{cc} A_2 & b_2 \\ b_2^T & c_2 \end{array} \right] \right) \leq 0$$

is feasible. From §B.3 this is equivalent to feasibility of

$$\left[\begin{array}{c} v \\ w \end{array} \right]^T \left[\begin{array}{cc} A_1 & b_1 \\ b_1^T & c_1 \end{array} \right] \left[\begin{array}{c} v \\ w \end{array} \right] < 0, \qquad \left[\begin{array}{c} v \\ w \end{array} \right]^T \left[\begin{array}{cc} A_2 & b_2 \\ b_2^T & c_2 \end{array} \right] \left[\begin{array}{c} v \\ w \end{array} \right] \leq 0.$$

If $w \neq 0$, then $x = v/w$ is feasible in (B.6). If $w = 0$, we have $v^T A_1 v < 0$, $v^T A_2 v \leq 0$, so $x = \hat{x} + tv$ satisfies

$$\begin{array}{rcl} x^T A_1 x + 2b_1^T x + c_1 & = & \hat{x}^T A_1 \hat{x} + 2b_1^T \hat{x} + c_1 + t^2 v^T A_1 v + 2t(A_1 \hat{x} + b_1)^T v \\ x^T A_2 x + 2b_2^T x + c_2 & = & \hat{x}^T A_2 \hat{x} + 2b_2^T \hat{x} + c_2 + t^2 v^T A_2 v + 2t(A_2 \hat{x} + b_2)^T v \\ & < & 2t(A_2 \hat{x} + b_2)^T v, \end{array}$$

i.e., x becomes feasible as $t \to \pm\infty$, depending on the sign of $(A_2\hat{x} + b_2)^T v$.

Finally, we prove the result in §B.1, *i.e.*, that the optimal values of (B.1) and (B.2) are equal if (B.1) is strictly feasible. To do this we note that γ is a lower bound for the optimal value of (B.1) if

$$x^T A_1 x + b_1^T x + c_1 \leq 0 \quad \Longrightarrow \quad x^T A_0 x + b_0^T x + c_0 \geq \gamma.$$

By the S-procedure this is true if and only if there exists a $\lambda \geq 0$ such that

$$\left[\begin{array}{cc} A_0 & b_0 \\ b_0^T & c_0 - \gamma \end{array} \right] + \lambda \left[\begin{array}{cc} A_1 & b_1 \\ b_1^T & c_1 \end{array} \right] \succeq 0,$$

i.e., γ, λ are feasible in (B.2).

Bibliography

The results in this appendix are known under different names in different disciplines. The term S-procedure is from control; see Boyd, El Ghaoui, Feron, and Balakrishnan [BEFB94, pages 23, 33] for a survey and references. Variations of the S-procedure are known in linear algebra in the context of joint diagonalization of a pair of symmetric matrices; see, for example, Calabi [Cal64] and Uhlig [Uhl79]. Special cases of the strong duality result are studied in the nonlinear programming literature on trust-region methods (Stern and Wolkowicz [SW95], Nocedal and Wright [NW99, page 78]).

Brickman [Bri61] proves that the field of values of a pair of matrices $A, B \in \mathbf{S}^n$ (*i.e.*, the set $F(A, B)$ defined in remark B.1) is a convex set if $n > 2$, and that the set $W(A, B)$ is a convex cone (for any n). Our proof in §B.3 is based on Hestenes [Hes68]. Many related results and additional references can be found in Horn and Johnson [HJ91, §1.8] and Ben-Tal and Nemirovski [BTN01, §4.10.5].

Appendix C

Numerical linear algebra background

In this appendix we give a brief overview of some basic numerical linear algebra, concentrating on methods for solving one or more sets of linear equations. We focus on direct (*i.e.*, noniterative) methods, and how problem structure can be exploited to improve efficiency. There are many important issues and methods in numerical linear algebra that we do not consider here, including numerical stability, details of matrix factorizations, methods for parallel or multiple processors, and iterative methods. For these (and other) topics, we refer the reader to the references given at the end of this appendix.

C.1 Matrix structure and algorithm complexity

We concentrate on methods for solving the set of linear equations

$$Ax = b \tag{C.1}$$

where $A \in \mathbf{R}^{n\times n}$ and $b \in \mathbf{R}^n$. We assume A is nonsingular, so the solution is unique for all values of b, and given by $x = A^{-1}b$. This basic problem arises in many optimization algorithms, and often accounts for most of the computation. In the context of solving the linear equations (C.1), the matrix A is often called the *coefficient matrix*, and the vector b is called the *righthand side*.

The standard generic methods for solving (C.1) require a computational effort that grows approximately like n^3. These methods assume nothing more about A than nonsingularity, and so are generally applicable. For n several hundred or smaller, these generic methods are probably the best methods to use, except in the most demanding real-time applications. For n more than a thousand or so, the generic methods of solving $Ax = b$ become less practical.

Coefficient matrix structure

In many cases the coefficient matrix A has some special structure or form that can be exploited to solve the equation $Ax = b$ more efficiently, using methods tailored for the special structure. For example, in the Newton system $\nabla^2 f(x)\Delta x_{\rm nt} = -\nabla f(x)$, the coefficient matrix is symmetric and positive definite, which allows us to use a solution method that is around twice as fast as the generic method (and also has better roundoff properties). There are many other types of structure that can be exploited, with computational savings (or algorithm speedup) that is usually far more than a factor of two. In many cases, the effort is reduced to something proportional to n^2 or even n, as compared to n^3 for the generic methods. Since these methods are usually applied when n is at least a hundred, and often far larger, the savings can be dramatic.

A wide variety of coefficient matrix structures can be exploited. Simple examples related to the sparsity pattern (*i.e.*, the pattern of zero and nonzero entries in the matrix) include banded, block diagonal, or sparse matrices. A more subtle exploitable structure is diagonal plus low rank. Many common forms of convex optimization problems lead to linear equations with coefficient matrices that have these exploitable structures. (There are many other matrix structures that can be exploited, *e.g.*, Toeplitz, Hankel, and circulant, that we will not consider in this appendix.)

We refer to a generic method that does not exploit any sparsity pattern in the matrices as one for *dense matrices*. We refer to a method that does not exploit any structure at all in the matrices as one for *unstructured matrices*.

C.1.1 Complexity analysis via flop count

The cost of a numerical linear algebra algorithm is often expressed by giving the total number of *floating-point operations* or *flops* required to carry it out, as a function of various problem dimensions. We define a flop as one addition, subtraction, multiplication, or division of two floating-point numbers. (Some authors define a flop as one multiplication followed by one addition, so their flop counts are smaller by a factor up to two.) To evaluate the complexity of an algorithm, we count the total number of flops, express it as a function (usually a polynomial) of the dimensions of the matrices and vectors involved, and simplify the expression by ignoring all terms except the leading (*i.e.*, highest order or dominant) terms.

As an example, suppose that a particular algorithm requires a total of

$$m^3 + 3m^2n + mn + 4mn^2 + 5m + 22$$

flops, where m and n are problem dimensions. We would normally simplify this flop count to

$$m^3 + 3m^2n + 4mn^2$$

flops, since these are the leading terms in the problem dimensions m and n. If in addition we assumed that $m \ll n$, we would further simplify the flop count to $4mn^2$.

Flop counts were originally popularized when floating-point operations were relatively slow, so counting the number gave a good estimate of the total computation time. This is no longer the case: Issues such as cache boundaries and locality of reference can dramatically affect the computation time of a numerical algorithm. However, flop counts can still give us a good rough estimate of the computation time of a numerical algorithm, and how the time grows with increasing problem size. Since a flop count no longer accurately predicts the computation time of an algorithm, we usually pay most attention to its order or orders, *i.e.*, its largest exponents, and ignore differences in flop counts smaller than a factor of two or so. For example, an algorithm with flop count $5n^2$ is considered comparable to one with a flop count $4n^2$, but faster than an algorithm with flop count $(1/3)n^3$.

C.1.2 Cost of basic matrix-vector operations

Vector operations

To compute the inner product $x^T y$ of two vectors $x, y \in \mathbf{R}^n$ we form the products $x_i y_i$, and then add them, which requires n multiplies and $n-1$ additions, or $2n-1$ flops. As mentioned above, we keep only the leading term, and say that the inner product requires $2n$ flops, or even more approximately, order n flops. A scalar-vector multiplication αx, where $\alpha \in \mathbf{R}$ and $x \in \mathbf{R}^n$ costs n flops. The addition $x + y$ of two vectors $x, y \in \mathbf{R}^n$ also costs n flops.

If the vectors x and y are sparse, *i.e.*, have only a few nonzero terms, these basic operations can be carried out faster (assuming the vectors are stored using an appropriate data structure). For example, if x is a sparse vector with N nonzero entries, then the inner product $x^T y$ can be computed in $2N$ flops.

Matrix-vector multiplication

A matrix-vector multiplication $y = Ax$ where $A \in \mathbf{R}^{m \times n}$ costs $2mn$ flops: We have to calculate m components of y, each of which is the product of a row of A with x, *i.e.*, an inner product of two vectors in $\mathbf{R}^n$.

Matrix-vector products can often be accelerated by taking advantage of structure in A. For example, if A is diagonal, then Ax can be computed in n flops, instead of $2n^2$ flops for multiplication by a general $n \times n$ matrix. More generally, if A is sparse, with only N nonzero elements (out of mn), then $2N$ flops are needed to form Ax, since we can skip multiplications and additions with zero.

As a less obvious example, suppose the matrix A has rank $p \ll \min\{m, n\}$, and is represented (stored) in the factored form $A = UV$, where $U \in \mathbf{R}^{m \times p}$, $V \in \mathbf{R}^{p \times n}$. Then we can compute Ax by first computing Vx (which costs $2pn$ flops), and then computing $U(Vx)$ (which costs $2mp$ flops), so the total is $2p(m + n)$ flops. Since $p \ll \min\{m, n\}$, this is small compared to $2mn$.

Matrix-matrix multiplication

The matrix-matrix product $C = AB$, where $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{n \times p}$, costs $2mnp$ flops. We have mp elements in C to calculate, each of which is an inner product of

two vectors of length n. Again, we can often make substantial savings by taking advantage of structure in A and B. For example, if A and B are sparse, we can accelerate the multiplication by skipping additions and multiplications with zero. If $m = p$ and we know that C is symmetric, then we can calculate the matrix product in m^2n flops, since we only have to compute the $(1/2)m(m+1)$ elements in the lower triangular part.

To form the product of several matrices, we can carry out the matrix-matrix multiplications in different ways, which have different flop counts in general. The simplest example is computing the product $D = ABC$, where $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$, and $C \in \mathbf{R}^{p\times q}$. Here we can compute D in two ways, using matrix-matrix multiplies. One method is to first form the product AB ($2mnp$ flops), and then form $D = (AB)C$ ($2mpq$ flops), so the total is $2mp(n+q)$ flops. Alternatively, we can first form the product BC ($2npq$ flops), and then form $D = A(BC)$ ($2mnq$ flops), with a total of $2nq(m+p)$ flops. The first method is better when $2mp(n+q) < 2nq(m+p)$, *i.e.*, when

$$\frac{1}{n} + \frac{1}{q} < \frac{1}{m} + \frac{1}{p}.$$

This assumes that no structure of the matrices is exploited in carrying out matrix-matrix products.

For products of more than three matrices, there are many ways to parse the product into matrix-matrix multiplications. Although it is not hard to develop an algorithm that determines the best parsing (*i.e.*, the one with the fewest required flops) given the matrix dimensions, in most applications the best parsing is clear.

C.2 Solving linear equations with factored matrices

C.2.1 Linear equations that are easy to solve

We start by examining some cases for which $Ax = b$ is easily solved, *i.e.*, $x = A^{-1}b$ is easily computed.

Diagonal matrices

Suppose A is diagonal and nonsingular (*i.e.*, $a_{ii} \neq 0$ for all i). The set of linear equations $Ax = b$ can be written as $a_{ii}x_i = b_i$, $i = 1, \ldots, n$. The solution is given by $x_i = b_i/a_{ii}$, and can be calculated in n flops.

Lower triangular matrices

A matrix $A \in \mathbf{R}^{n\times n}$ is *lower triangular* if $a_{ij} = 0$ for $j > i$. A lower triangular matrix is called *unit lower triangular* if the diagonal elements are equal to one. A lower triangular matrix is nonsingular if and only if $a_{ii} \neq 0$ for all i.

Suppose A is lower triangular and nonsingular. The equations $Ax = b$ are

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

From the first row, we have $a_{11}x_1 = b_1$, from which we conclude $x_1 = b_1/a_{11}$. From the second row we have $a_{21}x_1 + a_{22}x_2 = b_2$, so we can express x_2 as $x_2 = (b_2 - a_{21}x_1)/a_{22}$. (We have already computed x_1, so every number on the righthand side is known.) Continuing this way, we can express each component of x in terms of previous components, yielding the algorithm

$$\begin{aligned} x_1 &:= b_1/a_{11} \\ x_2 &:= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &:= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ &\vdots \\ x_n &:= (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{nn}. \end{aligned}$$

This procedure is called *forward substitution*, since we successively compute the components of x by substituting the known values into the next equation.

Let us give a flop count for forward substitution. We start by calculating x_1 (1 flop). We substitute x_1 in the second equation to find x_2 (3 flops), then substitute x_1 and x_2 in the third equation to find x_3 (5 flops), etc. The total number of flops is

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2.$$

Thus, when A is lower triangular and nonsingular, we can compute $x = A^{-1}b$ in n^2 flops.

If the matrix A has additional structure, in addition to being lower triangular, then forward substitution can be more efficient than n^2 flops. For example, if A is sparse (or banded), with at most k nonzero entries per row, then each forward substitution step requires at most $2k+1$ flops, so the overall flop count is $2(k+1)n$, or $2kn$ after dropping the term $2n$.

Upper triangular matrices

A matrix $A \in \mathbf{R}^{n\times n}$ is *upper triangular* if A^T is lower triangular, *i.e.*, if $a_{ij} = 0$ for $j < i$. We can solve linear equations with nonsingular upper triangular coefficient matrix in a way similar to forward substitution, except that we start by calculating x_n, then x_{n-1}, and so on. The algorithm is

$$\begin{aligned} x_n &:= b_n/a_{nn} \\ x_{n-1} &:= (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1} \\ x_{n-2} &:= (b_{n-2} - a_{n-2,n-1}x_{n-1} - a_{n-2,n}x_n)/a_{n-2,n-2} \\ &\vdots \\ x_1 &:= (b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)/a_{11}. \end{aligned}$$

This is called *backward substitution* or *back substitution* since we determine the coefficients in backward order. The cost to compute $x = A^{-1}b$ via backward substitution is n^2 flops. If A is upper triangular and sparse (or banded), with at most k nonzero entries per row, then back substitution costs $2kn$ flops.

Orthogonal matrices

A matrix $A \in \mathbf{R}^{n\times n}$ is *orthogonal* if $A^TA = I$, *i.e.*, $A^{-1} = A^T$. In this case we can compute $x = A^{-1}b$ by a simple matrix-vector product $x = A^Tb$, which costs $2n^2$ in general.

If the matrix A has additional structure, we can compute $x = A^{-1}b$ even more efficiently than $2n^2$ flops. For example, if A has the form $A = I - 2uu^T$, where $\|u\|_2 = 1$, we can compute

$$x = A^{-1}b = (I - 2uu^T)^Tb = b - 2(u^Tb)u$$

by first computing u^Tb, then forming $b - 2(u^Tb)u$, which costs $4n$ flops.

Permutation matrices

Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation of $(1, 2, \ldots, n)$. The associated *permutation matrix* $A \in \mathbf{R}^{n\times n}$ is given by

$$A_{ij} = \begin{cases} 1 & j = \pi_i \\ 0 & \text{otherwise.} \end{cases}$$

In each row (or column) of a permutation matrix there is exactly one entry with value one; all other entries are zero. Multiplying a vector by a permutation matrix simply permutes its coefficients:

$$Ax = (x_{\pi_1}, \ldots, x_{\pi_n}).$$

The inverse of a permutation matrix is the permutation matrix associated with the inverse permutation π^{-1}. This turns out to be A^T, which shows that permutation matrices are orthogonal.

If A is a permutation matrix, solving $Ax = b$ is very easy: x is obtained by permuting the entries of b by π^{-1}. This requires no floating point operations, according to our definition (but, depending on the implementation, might involve copying floating point numbers). We can reach the same conclusion from the equation $x = A^Tb$. The matrix A^T (like A) has only one nonzero entry per row, with value one. Thus no additions are required, and the only multiplications required are by one.

C.2.2 The factor-solve method

The basic approach to solving $Ax = b$ is based on expressing A as a product of nonsingular matrices,

$$A = A_1A_2 \cdots A_k,$$

so that

$$x = A^{-1}b = A_k^{-1}A_{k-1}^{-1}\cdots A_1^{-1}b.$$

We can compute x using this formula, working from right to left:

$$\begin{aligned} z_1 &:= A_1^{-1}b \\ z_2 &:= A_2^{-1}z_1 = A_2^{-1}A_1^{-1}b \\ &\vdots \\ z_{k-1} &:= A_{k-1}^{-1}z_{k-2} = A_{k-1}^{-1}\cdots A_1^{-1}b \\ x &:= A_k^{-1}z_{k-1} = A_k^{-1}\cdots A_1^{-1}b. \end{aligned}$$

The ith step of this process requires computing $z_i = A_i^{-1}z_{i-1}$, *i.e.*, solving the linear equations $A_i z_i = z_{i-1}$. If each of these equations is easy to solve (*e.g.*, if A_i is diagonal, lower or upper triangular, a permutation, etc.), this gives a method for computing $x = A^{-1}b$.

The step of expressing A in factored form (*i.e.*, computing the factors A_i) is called the *factorization step*, and the process of computing $x = A^{-1}b$ recursively, by solving a sequence problems of the form $A_i z_i = z_{i-1}$, is often called the *solve step*. The total flop count for solving $Ax = b$ using this factor-solve method is $f + s$, where f is the flop count for computing the factorization, and s is the total flop count for the solve step. In many cases, the cost of the factorization, f, dominates the total solve cost s. In this case, the cost of solving $Ax = b$, *i.e.*, computing $x = A^{-1}b$, is just f.

Solving equations with multiple righthand sides

Suppose we need to solve the equations

$$Ax_1 = b_1, \qquad Ax_2 = b_2, \qquad \ldots, \qquad Ax_m = b_m,$$

where $A \in \mathbf{R}^{n\times n}$ is nonsingular. In other words, we need to solve m sets of linear equations, with the same coefficient matrix, but different righthand sides. Alternatively, we can think of this as computing the matrix

$$X = A^{-1}B$$

where

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix} \in \mathbf{R}^{n\times m}, \qquad B = \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix} \in \mathbf{R}^{n\times m}.$$

To do this, we first factor A, which costs f. Then for $i = 1, \ldots, m$ we compute $A^{-1}b_i$ using the solve step. Since we only factor A once, the total effort is

$$f + ms.$$

In other words, we amortize the factorization cost over the set of m solves. Had we (needlessly) repeated the factorization step for each i, the cost would be $m(f + s)$.

When the factorization cost f dominates the solve cost s, the factor-solve method allows us to solve a small number of linear systems, with the same coefficient matrix, at essentially the same cost as solving one. This is because the most expensive step, the factorization, is done only once.

We can use the factor-solve method to compute the inverse A^{-1} by solving $Ax = e_i$ for $i = 1, \ldots, n$, *i.e.*, by computing $A^{-1}I$. This requires one factorization and n solves, so the cost is $f + ns$.

C.3 LU, Cholesky, and LDL^T factorization

C.3.1 LU factorization

Every nonsingular matrix $A \in \mathbf{R}^{n\times n}$ can be factored as

$$A = PLU$$

where $P \in \mathbf{R}^{n\times n}$ is a permutation matrix, $L \in \mathbf{R}^{n\times n}$ is unit lower triangular, and $U \in \mathbf{R}^{n\times n}$ is upper triangular and nonsingular. This is called the *LU factorization* of A. We can also write the factorization as $P^TA = LU$, where the matrix P^TA is obtained from A by re-ordering the rows. The standard algorithm for computing an LU factorization is called *Gaussian elimination with partial pivoting* or *Gaussian elimination with row pivoting*. The cost is $(2/3)n^3$ flops if no structure in A is exploited, which is the case we consider first.

Solving sets of linear equations using the LU factorization

The LU factorization, combined with the factor-solve approach, is the standard method for solving a general set of linear equations $Ax = b$.

Algorithm C.1 *Solving linear equations by LU factorization.*

given a set of linear equations $Ax = b$, with A nonsingular.

1. *LU factorization.* Factor A as $A = PLU$ ($(2/3)n^3$ flops).
2. *Permutation.* Solve $Pz_1 = b$ (0 flops).
3. *Forward substitution.* Solve $Lz_2 = z_1$ (n^2 flops).
4. *Backward substitution.* Solve $Ux = z_2$ (n^2 flops).

The total cost is $(2/3)n^3 + 2n^2$, or $(2/3)n^3$ flops if we keep only the leading term.

If we need to solve multiple sets of linear equations with different righthand sides, *i.e.*, $Ax_i = b_i$, $i = 1, \ldots, m$, the cost is

$$(2/3)n^3 + 2mn^2,$$

since we factor A once, and carry out m pairs of forward and backward substitutions. For example, we can solve two sets of linear equations, with the same coefficient matrix but different righthand sides, at essentially the same cost as solving one. We can compute the inverse A^{-1} by solving the equations $Ax_i = e_i$, where x_i is the ith column of A^{-1}, and e_i is the ith unit vector. This costs $(8/3)n^3$, *i.e.*, about $3n^3$ flops.

If the matrix A has certain structure, for example banded or sparse, the LU factorization can be computed in less than $(2/3)n^3$ flops, and the associated forward and backward substitutions can also be carried out more efficiently.

LU factorization of banded matrices

Suppose the matrix $A \in \mathbf{R}^{n\times n}$ is *banded*, *i.e.*, $a_{ij} = 0$ if $|i-j| > k$, where $k < n-1$ is called the *bandwidth* of A. We are interested in the case where $k \ll n$, *i.e.*, the bandwidth is much smaller than the size of the matrix. In this case an LU factorization of A can be computed in roughly $4nk^2$ flops. The resulting upper triangular matrix U has bandwidth at most $2k$, and the lower triangular matrix L has at most $k+1$ nonzeros per column, so the forward and back substitutions can be carried out in order $6nk$ flops. Therefore if A is banded, the linear equations $Ax = b$ can be solved in about $4nk^2$ flops.

LU factorization of sparse matrices

When the matrix A is sparse, the LU factorization usually includes both row and column permutations, *i.e.*, A is factored as

$$A = P_1 L U P_2,$$

where P_1 and P_2 are permutation matrices, L is lower triangular, and U is upper triangular. If the factors L and U are sparse, the forward and backward substitutions can be carried out efficiently, and we have an efficient method for solving $Ax = b$. The sparsity of the factors L and U depends on the permutations P_1 and P_2, which are chosen in part to yield relatively sparse factors.

The cost of computing the sparse LU factorization depends in a complicated way on the size of A, the number of nonzero elements, its sparsity pattern, and the particular algorithm used, but is often dramatically smaller than the cost of a dense LU factorization. In many cases the cost grows approximately linearly with n, when n is large. This means that when A is sparse, we can solve $Ax = b$ very efficiently, often with an order approximately n.

C.3.2 Cholesky factorization

If $A \in \mathbf{R}^{n\times n}$ is symmetric and positive definite, then it can be factored as

$$A = LL^T$$

where L is lower triangular and nonsingular with positive diagonal elements. This is called the *Cholesky factorization* of A, and can be interpreted as a symmetric LU factorization (with $L = U^T$). The matrix L, which is uniquely determined by A, is called the *Cholesky factor* of A. The cost of computing the Cholesky factorization of a dense matrix, *i.e.*, without exploiting any structure, is $(1/3)n^3$ flops, half the cost of an LU factorization.

Solving positive definite sets of equations using Cholesky factorization

The Cholesky factorization can be used to solve $Ax = b$ when A is symmetric positive definite.

Algorithm C.2 *Solving linear equations by Cholesky factorization.*

given a set of linear equations $Ax = b$, with $A \in \mathbf{S}^n_{++}$.

1. *Cholesky factorization.* Factor A as $A = LL^T$ ($(1/3)n^3$ flops).
2. *Forward substitution.* Solve $Lz_1 = b$ (n^2 flops).
3. *Backward substitution.* Solve $L^T x = z_1$ (n^2 flops).

The total cost is $(1/3)n^3 + 2n^2$, or roughly $(1/3)n^3$ flops.

There are specialized algorithms, with a complexity much lower than $(1/3)n^3$, for Cholesky factorization of banded and sparse matrices.

Cholesky factorization of banded matrices

If A is symmetric positive definite and banded with bandwidth k, then its Cholesky factor L is banded with bandwidth k, and can be calculated in nk^2 flops. The cost of the associated solve step is $4nk$ flops.

Cholesky factorization of sparse matrices

When A is symmetric positive definite and sparse, it is usually factored as

$$A = PLL^T P^T,$$

where P is a permutation matrix and L is lower triangular with positive diagonal elements. We can also express this as $P^T AP = LL^T$, *i.e.*, LL^T is the Cholesky factorization of $P^T AP$. We can interpret this as first re-ordering the variables and equations, and then forming the (standard) Cholesky factorization of the resulting permuted matrix. Since $P^T AP$ is positive definite for any permutation matrix P, we are free to choose any permutation matrix; for each choice there is a unique associated Cholesky factor L. The choice of P, however, can greatly affect the sparsity of the factor L, which in turn can greatly affect the efficiency of solving $Ax = b$. Various heuristic methods are used to select a permutation P that leads to a sparse factor L.

Example C.1 *Cholesky factorization with an arrow sparsity pattern.* Consider a sparse matrix of the form

$$A = \begin{bmatrix} 1 & u^T \\ u & D \end{bmatrix}$$

where $D \in \mathbf{R}^{n\times n}$ is positive diagonal, and $u \in \mathbf{R}^n$. It can be shown that A is positive definite if $u^T D^{-1} u < 1$. The Cholesky factorization of A is

$$\begin{bmatrix} 1 & u^T \\ u & D \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ u & L \end{bmatrix} \begin{bmatrix} 1 & u^T \\ 0 & L^T \end{bmatrix} \tag{C.2}$$

where L is lower triangular with $LL^T = D - uu^T$. For general u, the matrix $D - uu^T$ is dense, so we can expect L to be dense. Although the matrix A is very sparse (most of its rows have just two nonzero elements), its Cholesky factors are almost completely dense.

On the other hand, suppose we permute the first row and column of A to the end. After this re-ordering, we obtain the Cholesky factorization

$$\begin{bmatrix} D & u \\ u^T & 1 \end{bmatrix} = \begin{bmatrix} D^{1/2} & 0 \\ u^T D^{-1/2} & \sqrt{1-u^T D^{-1} u} \end{bmatrix} \begin{bmatrix} D^{1/2} & D^{-1/2}u^T \\ 0 & \sqrt{1-u^T D^{-1} u} \end{bmatrix}.$$

Now the Cholesky factor has a diagonal 1,1 block, so it is very sparse.

This example illustrates that the re-ordering greatly affects the sparsity of the Cholesky factors. Here it was quite obvious what the best permutation is, and all good re-ordering heuristics would select this re-ordering and permute the dense row and column to the end. For more complicated sparsity patterns, it can be very difficult to find the 'best' re-ordering (*i.e.*, resulting in the greatest number of zero elements in L), but various heuristics provide good suboptimal permutations.

For the sparse Cholesky factorization, the re-ordering permutation P is often determined using only sparsity pattern of the matrix A, and not the particular numerical values of the nonzero elements of A. Once P is chosen, we can also determine the sparsity pattern of L without knowing the numerical values of the nonzero entries of A. These two steps combined are called the *symbolic factorization* of A, and form the first step in a sparse Cholesky factorization. In contrast, the permutation matrices in a sparse LU factorization do depend on the numerical values in A, in addition to its sparsity pattern.

The symbolic factorization is then followed by the *numerical factorization*, *i.e.*, the calculation of the nonzero elements of L. Software packages for sparse Cholesky factorization often include separate routines for the symbolic and the numerical factorization. This is useful in many applications, because the cost of the symbolic factorization is significant, and often comparable to the numerical factorization. Suppose, for example, that we need to solve m sets of linear equations

$$A_1 x = b_1, \qquad A_2 x = b_2, \qquad \ldots, \qquad A_m x = b_m$$

where the matrices A_i are symmetric positive definite, with different numerical values, but the same sparsity pattern. Suppose the cost of a symbolic factorization is f_{symb}, the cost of a numerical factorization is f_{num}, and the cost of the solve step is s. Then we can solve the m sets of linear equations in

$$f_{\text{symb}} + m(f_{\text{num}} + s)$$

flops, since we only need to carry out the symbolic factorization once, for all m sets of equations. If instead we carry out a separate symbolic factorization for each set of linear equations, the flop count is $m(f_{\text{symb}} + f_{\text{num}} + s)$.

C.3.3 LDL^T factorization

Every nonsingular symmetric matrix A can be factored as

$$A = PLDL^T P^T$$

where P is a permutation matrix, L is lower triangular with positive diagonal elements, and D is block diagonal, with nonsingular 1×1 and 2×2 diagonal blocks. This is called an $\mathrm{LDL}^{\mathrm{T}}$ factorization of A. (The Cholesky factorization can be considered a special case of $\mathrm{LDL}^{\mathrm{T}}$ factorization, with $P = I$ and $D = I$.) An $\mathrm{LDL}^{\mathrm{T}}$ factorization can be computed in $(1/3)n^3$ flops, if no structure of A is exploited.

Algorithm C.3 *Solving linear equations by LDL^T factorization.*

given a set of linear equations $Ax = b$, with $A \in \mathbf{S}^n$ nonsingular.

1. *LDL^T factorization.* Factor A as $A = PLDL^TP$ ($(1/3)n^3$ flops).
2. *Permutation.* Solve $Pz_1 = b$ (0 flops).
3. *Forward substitution.* Solve $Lz_2 = z_1$ (n^2 flops).
4. *(Block) diagonal solve.* Solve $Dz_3 = z_2$ (order n flops).
5. *Backward substitution.* Solve $L^Tz_4 = z_3$ (n^2 flops).
6. *Permutation.* Solve $P^Tx = z_4$ (0 flops).

The total cost is, keeping only the dominant term, $(1/3)n^3$ flops.

$\mathrm{LDL}^{\mathrm{T}}$ factorization of banded and sparse matrices

As with the LU and Cholesky factorizations, there are specialized methods for calculating the $\mathrm{LDL}^{\mathrm{T}}$ factorization of a sparse or banded matrix. These are similar to the analogous methods for Cholesky factorization, with the additional factor D. In a sparse $\mathrm{LDL}^{\mathrm{T}}$ factorization, the permutation matrix P cannot be chosen only on the basis of the sparsity pattern of A (as in a sparse Cholesky factorization); it also depends on the particular nonzero values in the matrix A.

C.4 Block elimination and Schur complements

C.4.1 Eliminating a block of variables

In this section we describe a general method that can be used to solve $Ax = b$ by first eliminating a subset of the variables, and then solving a smaller system of linear equations for the remaining variables. For a dense unstructured matrix, this approach gives no advantage. But when the submatrix of A associated with the eliminated variables is easily factored (for example, if it is block diagonal or banded) the method can be substantially more efficient than a general method.

Suppose we partition the variable $x \in \mathbf{R}^n$ into two blocks or subvectors,

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

where $x_1 \in \mathbf{R}^{n_1}$, $x_2 \in \mathbf{R}^{n_2}$. We conformally partition the linear equations $Ax = b$ as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{C.3}$$

where $A_{11} \in \mathbf{R}^{n_1 \times n_1}$, $A_{22} \in \mathbf{R}^{n_2 \times n_2}$. Assuming that the submatrix A_{11} is invertible, we can eliminate x_1 from the equations, as follows. Using the first equation, we can express x_1 in terms of x_2:

$$x_1 = A_{11}^{-1}(b_1 - A_{12}x_2). \tag{C.4}$$

Substituting this expression into the second equation yields

$$(A_{22} - A_{21}A_{11}^{-1}A_{12})x_2 = b_2 - A_{21}A_{11}^{-1}b_1. \tag{C.5}$$

We refer to this as the *reduced equation* obtained by eliminating x_1 from the original equation. The reduced equation (C.5) and the equation (C.4) together are equivalent to the original equations (C.3). The matrix appearing in the reduced equation is called the *Schur complement* of the first block A_{11} in A:

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

(see also §A.5.5). The Schur complement S is nonsingular if and only if A is nonsingular.

The two equations (C.5) and (C.4) give us an alternative approach to solving the original system of equations (C.3). We first form the Schur complement S, then find x_2 by solving (C.5), and then calculate x_1 from (C.4). We can summarize this method as follows.

Algorithm C.4 *Solving linear equations by block elimination.*

given a nonsingular set of linear equations (C.3), with A_{11} nonsingular.

1. Form $A_{11}^{-1}A_{12}$ and $A_{11}^{-1}b_1$.
2. Form $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ and $\tilde{b} = b_2 - A_{21}A_{11}^{-1}b_1$.
3. Determine x_2 by solving $Sx_2 = \tilde{b}$.
4. Determine x_1 by solving $A_{11}x_1 = b_1 - A_{12}x_2$.

Remark C.1 *Interpretation as block factor-solve.* Block elimination can be interpreted in terms of the factor-solve approach described in §C.2.2, based on the factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix},$$

which can be considered a block LU factorization. This block LU factorization suggests the following method for solving (C.3). We first do a 'block forward substitution' to solve

$$\begin{bmatrix} A_{11} & 0 \\ A_{21} & S \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

and then solve

$$\begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

by 'block backward substitution'. This yields the same expressions as the block elimination method:

$$\begin{aligned} z_1 &= A_{11}^{-1} b_1 \\ z_2 &= S^{-1}(b_2 - A_{21} z_1) \\ x_2 &= z_2 \\ x_1 &= z_1 - A_{11}^{-1} A_{12} z_2. \end{aligned}$$

In fact, the modern approach to the factor-solve method is based on block factor and solve steps like these, with the block sizes optimally chosen for the processor (or processors), cache sizes, etc.

Complexity analysis of block elimination method

To analyze the (possible) advantage of solving the set of linear equations using block elimination, we carry out a flop count. We let f and s denote the cost of factoring A_{11} and carrying out the associated solve step, respectively. To keep the analysis simple we assume (for now) that A_{12}, A_{22}, and A_{21} are treated as dense, unstructured matrices. The flop counts for each of the four steps in solving $Ax = b$ using block elimination are:

1. Computing $A_{11}^{-1}A_{12}$ and $A_{11}^{-1}b_1$ requires factoring A_{11} and $n_2 + 1$ solves, so it costs $f + (n_2 + 1)s$, or just $f + n_2 s$, dropping the dominated term s.

2. Forming the Schur complement S requires the matrix multiply $A_{21}(A_{11}^{-1}A_{12})$, which costs $2n_2^2 n_1$, and an $n_2 \times n_2$ matrix subtraction, which costs n_2^2 (and can be dropped). The cost of forming $\tilde{b} = b_2 - A_{21}A_{11}^{-1}b_1$ is dominated by the cost of forming S, and so can be ignored. The total cost of step 2, ignoring dominated terms, is then $2n_2^2 n_1$.

3. To compute $x_2 = S^{-1}\tilde{b}$, we factor S and solve, which costs $(2/3)n_2^3$.

4. Forming $b_1 - A_{12}x_2$ costs $2n_1 n_2 + n_1$ flops. To compute $x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$, we can use the factorization of A_{11} already computed in step 1, so only the solve is necessary, which costs s. Both of these costs are dominated by other terms, and can be ignored.

The total cost is then

$$f + n_2 s + 2n_2^2 n_1 + (2/3)n_2^3 \tag{C.6}$$

flops.

Eliminating an unstructured matrix

We first consider the case when no structure in A_{11} is exploited. We factor A_{11} using a standard LU factorization, so $f = (2/3)n_1^3$, and then solve using a forward and a backward substitution, so $s = 2n_1^2$. The flop count for solving the equations via block elimination is then

$$(2/3)n_1^3 + n_2(2n_1^2) + 2n_2^2 n_1 + (2/3)n_2^3 = (2/3)(n_1 + n_2)^3,$$

which is the same as just solving the larger set of equations using a standard LU factorization. In other words, solving a set of equations by block elimination gives no advantage when no structure of A_{11} is exploited.

On the other hand, when the structure of A_{11} allows us to factor and solve more efficiently than the standard method, block elimination can be more efficient than applying the standard method.

Eliminating a diagonal matrix

If A_{11} is diagonal, no factorization is needed, and we can carry out a solve in n_1 flops, so we have $f = 0$ and $s = n_1$. Substituting these values into (C.6) and keeping only the leading terms yields

$$2n_2^2 n_1 + (2/3)n_2^3,$$

flops, which is far smaller than $(2/3)(n_1+n_2)^3$, the cost using the standard method. In particular, the flop count of the standard method grows cubicly in n_1, whereas for block elimination the flop count grows only linearly in n_1.

Eliminating a banded matrix

If A_{11} is banded with bandwidth k, we can carry out the factorization in about $f = 4k^2 n_1$ flops, and the solve can be done in about $s = 6kn_1$ flops. The overall complexity of solving $Ax = b$ using block elimination is

$$4k^2 n_1 + 6n_2 k n_1 + 2n_2^2 n_1 + (2/3)n_2^3$$

flops. Assuming k is small compared to n_1 and n_2, this simplifies to $2n_2^2 n_1 + (2/3)n_2^3$, the same as when A_{11} is diagonal. In particular, the complexity grows linearly in n_1, as opposed to cubicly in n_1 for the standard method.

A matrix for which A_{11} is banded is sometimes called an *arrow matrix* since the sparsity pattern, when $n_1 \gg n_2$, looks like an arrow pointing down and right. Block elimination can solve linear equations with arrow structure far more efficiently than the standard method.

Eliminating a block diagonal matrix

Suppose that A_{11} is block diagonal, with (square) block sizes $m_1, \ldots, m_k$, where $n_1 = m_1 + \cdots + m_k$. In this case we can factor A_{11} by factoring each block separately, and similarly we can carry out the solve step on each block separately. Using standard methods for these we find

$$f = (2/3)m_1^3 + \cdots + (2/3)m_k^3, \qquad s = 2m_1^2 + \cdots + 2m_k^2,$$

so the overall complexity of block elimination is

$$(2/3)\sum_{i=1}^{k} m_i^3 + 2n_2 \sum_{i=1}^{k} m_i^2 + 2n_2^2 \sum_{i=1}^{k} m_i + (2/3)n_2^3.$$

If the block sizes are small compared to n_1 and $n_1 \gg n_2$, the savings obtained by block elimination is dramatic.

The linear equations $Ax = b$, where A_{11} is block diagonal, are called *partially separable* for the following reason. If the subvector x_2 is fixed, the remaining equations decouple into k sets of independent linear equations (which can be solved separately). The subvector x_2 is sometimes called the *complicating variable* since the equations are much simpler when x_2 is fixed. Using block elimination, we can solve partially separable linear equations far more efficiently than by using a standard method.

Eliminating a sparse matrix

If A_{11} is sparse, we can eliminate A_{11} using a sparse factorization and sparse solve steps, so the values of f and s in (C.6) are much less than for unstructured A_{11}. When A_{11} in (C.3) is sparse and the other blocks are dense, and $n_2 \ll n_1$, we say that A is a sparse matrix with a few dense rows and columns. Eliminating the sparse block A_{11} provides an efficient method for solving equations which are sparse except for a few dense rows and columns.

An alternative is to simply apply a sparse factorization algorithm to the entire matrix A. Most sparse solvers will handle dense rows and columns, and select a permutation that results in sparse factors, and hence fast factorization and solve times. This is more straightforward than using block elimination, but often slower, especially in applications where we can exploit structure in the other blocks (see, *e.g.*, example C.4).

Remark C.2 As already suggested in remark C.1, these two methods for solving systems with a few dense rows and columns are closely related. Applying the elimination method by factoring A_{11} and S as

$$A_{11} = P_1 L_1 U_1 P_2, \qquad S = P_3 L_2 U_2,$$

can be interpreted as factoring A as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} P_1 & 0 \\ 0 & P_3 \end{bmatrix} \begin{bmatrix} L_1 & 0 \\ P_3^T A_{21} P_2^T U_1^{-1} & L_2 \end{bmatrix} \begin{bmatrix} U_1 & L_1^{-1} P_1^T A_{12} \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} P_2 & 0 \\ 0 & I \end{bmatrix},$$

followed by forward and backward substitutions.

C.4.2 Block elimination and structure

Symmetry and positive definiteness

There are variants of the block elimination method that can be used when A is symmetric, or symmetric and positive definite. When A is symmetric, so are A_{11} and the Schur complement S, so a symmetric factorization can be used for A_{11} and S. Symmetry can also be exploited in the other operations, such as the matrix multiplies. Overall the savings over the nonsymmetric case is around a factor of two.

Positive definiteness can also be exploited in block elimination. When A is symmetric and positive definite, so are A_{11} and the Schur complement S, so Cholesky factorizations can be used.

Exploiting structure in other blocks

Our complexity analysis above assumes that we exploit no structure in the matrices A_{12}, A_{21}, A_{22}, and the Schur complement S, *i.e.*, they are treated as dense. But in many cases there is structure in these blocks that can be exploited in forming the Schur complement, factoring it, and carrying out the solve steps. In such cases the computational savings of the block elimination method over a standard method can be even higher.

Example C.2 *Block triangular equations.* Suppose that $A_{12} = 0$, *i.e.*, the linear equations $Ax = b$ have block lower triangular structure:

$$\begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

In this case the Schur complement is just $S = A_{22}$, and the block elimination method reduces to block forward substitution:

$$\begin{array}{lcl} x_1 & := & A_{11}^{-1} b_1 \\ x_2 & := & A_{22}^{-1}(b_2 - A_{21}x_1). \end{array}$$

Example C.3 *Block diagonal and banded systems.* Suppose that A_{11} is block diagonal, with maximum block size $l \times l$, and that A_{12}, A_{21}, and A_{22} are banded, say with bandwidth k. In this case, A_{11}^{-1} is also block diagonal, with the same block sizes as A_{11}. Therefore the product $A_{11}^{-1}A_{21}$ is also banded, with bandwidth $k + l$, and the Schur complement, $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is banded with bandwidth $2k + l$. This means that forming the Schur complement S can be done more efficiently, and that the factorization and solve steps with S can be done efficiently. In particular, for fixed maximum block size l and bandwidth k, we can solve $Ax = b$ with a number of flops that grows linearly with n.

Example C.4 *KKT structure.* Suppose that the matrix A has *KKT structure*, *i.e.*,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & 0 \end{bmatrix},$$

where $A_{11} \in \mathbf{S}^p_{++}$, and $A_{12} \in \mathbf{R}^{p \times m}$ with $\mathbf{rank}\, A_{12} = m$. Since $A_{11} \succ 0$, we can use a Cholesky factorization. The Schur complement $S = -A_{12}^T A_{11}^{-1} A_{12}$ is negative definite, so we can factor $-S$ using a Cholesky factorization.

C.4.3 The matrix inversion lemma

The idea of block elimination is to remove variables, and then solve a smaller set of equations that involve the Schur complement of the original matrix with respect to the eliminated variables. The same idea can be turned around: When we recognize a matrix as a Schur complement, we can introduce new variables, and create a larger set of equations to solve. In most cases there is no advantage to doing this, since we end up with a larger set of equations. But when the larger set of equations has some special structure that can be exploited to solve it, introducing variables can lead to an efficient method. The most common case is when another block of variables can be eliminated from the larger matrix.

We start with the linear equations

$$(A + BC)x = b, \tag{C.7}$$

where $A \in \mathbf{R}^{n\times n}$ is nonsingular, and $B \in \mathbf{R}^{n\times p}$, $C \in \mathbf{R}^{p\times n}$. We introduce a new variable $y = Cx$, and rewrite the equations as

$$Ax + By = b, \qquad y = Cx,$$

or, in matrix form,

$$\begin{bmatrix} A & B \\ C & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{C.8}$$

Note that our original coefficient matrix, $A + BC$, is the Schur complement of $-I$ in the larger matrix that appears in (C.8). If we were to eliminate the variable y from (C.8), we would get back the original equation (C.7).

In some cases, it can be more efficient to solve the larger set of equations (C.8) than the original, smaller set of equations (C.7). This would be the case, for example, if A, B, and C were relatively sparse, but the matrix $A + BC$ were far less sparse.

After introducing the new variable y, we can eliminate the original variable x from the larger set of equations (C.8), using $x = A^{-1}(b - By)$. Substituting this into the second equation $y = Cx$, we obtain

$$(I + CA^{-1}B)y = CA^{-1}b,$$

so that

$$y = (I + CA^{-1}B)^{-1}CA^{-1}b.$$

Using $x = A^{-1}(b - By)$, we get

$$x = \left(A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1}\right) b. \tag{C.9}$$

Since b is arbitrary, we conclude that

$$(A + BC)^{-1} = A^{-1} - A^{-1}B\left(I + CA^{-1}B\right)^{-1}CA^{-1}.$$

This is known as the *matrix inversion lemma*, or the *Sherman-Woodbury-Morrison formula*.

The matrix inversion lemma has many applications. For example if p is small (or even just not very large), it gives us a method for solving $(A + BC)x = b$, provided we have an efficient method for solving $Au = v$.

Diagonal or sparse plus low rank

Suppose that A is diagonal with nonzero diagonal elements, and we want to solve an equation of the form (C.7). The straightforward solution would consist in first forming the matrix $D = A + BC$, and then solving $Dx = b$. If the product BC is dense, then the complexity of this method is $2pn^2$ flops to form $A + BC$, plus $(2/3)n^3$ flops for the LU factorization of D, so the total cost is

$$2pn^2 + (2/3)n^3$$

flops. The matrix inversion lemma suggests a more efficient method. We can calculate x by evaluating the expression (C.9) from right to left, as follows. We first evaluate $z = A^{-1}b$ (n flops, since A is diagonal). Then we form the matrix $E = I + CA^{-1}B$ ($2p^2n$ flops). Next we solve $Ew = Cz$, which is a set of p linear equations in p variables. The cost is $(2/3)p^3$ flops, plus $2pn$ to form Cz. Finally, we evaluate $x = z - A^{-1}Bw$ ($2pn$ flops for the matrix-vector product Bw, plus lower order terms). The total cost is

$$2p^2n + (2/3)p^3$$

flops, dropping dominated terms. Comparing with the first method, we see that the second method is more efficient when $p < n$. In particular if p is small and fixed, the complexity grows linearly with n.

Another important application of the matrix inversion lemma occurs when A is sparse and nonsingular, and the matrices B and C are dense. Again we can compare two methods. The first method is to form the (dense) matrix $A + BC$, and to solve (C.7) using a dense LU factorization. The cost of this method is $2pn^2+(2/3)n^3$ flops. The second method is based on evaluating the expression (C.9), using a sparse LU factorization of A. Specifically, suppose that f is the cost of factoring A as $A = P_1LUP_2$, and s is the cost of solving the factored system $P_1LUP_2x = d$. We can evaluate (C.9) from right to left as follows. We first factor A, and solve $p + 1$ linear systems

$$Az = b, \qquad AD = B,$$

to find $z \in \mathbf{R}^n$, and $D \in \mathbf{R}^{n\times p}$. The cost is $f + (p + 1)s$ flops. Next, we form the matrix $E = I + CD$, and solve

$$Ew = Cz,$$

which is a set of p linear equations in p variables w. The cost of this step is $2p^2n + (2/3)p^3$ plus lower order terms. Finally, we evaluate $x = z - Dw$, at a cost of $2pn$ flops. This gives us a total cost of

$$f + ps + 2p^2n + (2/3)p^3$$

flops. If $f \ll (2/3)n^3$ and $s \ll 2n^2$, this is much lower than the complexity of the first method.

Remark C.3 *The augmented system approach.* A different approach to exploiting sparse plus low rank structure is to solve (C.8) directly using a sparse LU-solver. The system (C.8) is a set of $p + n$ linear equations in $p + n$ variables, and is sometimes

called the *augmented* system associated with (C.7). If A is very sparse and p is small, then solving the augmented system using a sparse solver can be much faster than solving the system (C.7) using a dense solver.

The augmented system approach is closely related to the method that we described above. Suppose

$$A = P_1 L U P_2$$

is a sparse LU factorization of A, and

$$I + CA^{-1}B = P_3 \tilde{L} \tilde{U}$$

is a dense LU factorization of $I + CA^{-1}B$. Then

$$\begin{bmatrix} A & B \\ C & -I \end{bmatrix} = \begin{bmatrix} P_1 & 0 \\ 0 & P_3 \end{bmatrix} \begin{bmatrix} L & 0 \\ P_3^T C P_2^T U^{-1} & -\tilde{L} \end{bmatrix} \begin{bmatrix} U & L^{-1} P_1^T B \\ 0 & \tilde{U} \end{bmatrix} \begin{bmatrix} P_2 & 0 \\ 0 & I \end{bmatrix}, \tag{C.10}$$

and this factorization can be used to solve the augmented system. It can be verified that this is equivalent to the method based on the matrix inversion lemma that we described above.

Of course, if we solve the augmented system using a sparse LU solver, we have no control over the permutations that are selected. The solver might choose a factorization different from (C.10), and more expensive to compute. In spite of this, the augmented system approach remains an attractive option. It is easier to implement than the method based on the matrix inversion lemma, and it is numerically more stable.

Low rank updates

Suppose $A \in \mathbf{R}^{n \times n}$ is nonsingular, $u, v \in \mathbf{R}^n$ with $1 + v^T A^{-1} u \neq 0$, and we want to solve two sets of linear equations

$$Ax = b, \qquad (A + uv^T)\tilde{x} = b.$$

The solution $\tilde{x}$ of the second system is called a *rank-one update* of x. The matrix inversion lemma allows us to calculate the rank-one update $\tilde{x}$ very cheaply, once we have computed x. We have

$$\begin{aligned} \tilde{x} &= (A + uv^T)^{-1} b \\ &= (A^{-1} - \frac{1}{1 + v^T A^{-1} u} A^{-1} u v^T A^{-1}) b \\ &= x - \frac{v^T x}{1 + v^T A^{-1} u} A^{-1} u. \end{aligned}$$

We can therefore solve both systems by factoring A, computing $x = A^{-1}b$ and $w = A^{-1}u$, and then evaluating

$$\tilde{x} = x - \frac{v^T x}{1 + v^T w} w.$$

The overall cost is $f + 2s$, as opposed to $2(f + s)$ if we were to solve for $\tilde{x}$ from scratch.

C.5 Solving underdetermined linear equations

To conclude this appendix, we mention a few important facts about *underdetermined* linear equations

$$Ax = b, \tag{C.11}$$

where $A \in \mathbf{R}^{p\times n}$ with $p < n$. We assume that $\mathbf{rank}\, A = p$, so there is at least one solution for all b.

In many applications it is sufficient to find just one particular solution $\hat{x}$. In other situations we might need a complete parametrization of all solutions as

$$\{x \mid Ax = b\} = \{Fz + \hat{x} \mid z \in \mathbf{R}^{n-p}\} \tag{C.12}$$

where F is a matrix whose columns form a basis for the nullspace of A.

Inverting a nonsingular submatrix of A

The solution of the underdetermined system is straightforward if a $p\times p$ nonsingular submatrix of A is known. We start by assuming that the first p columns of A are independent. Then we can write the equation $Ax = b$ as

$$Ax = \begin{bmatrix} A_1 & A_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = A_1x_1 + A_2x_2 = b,$$

where $A_1 \in \mathbf{R}^{p\times p}$ is nonsingular. We can express x_1 as

$$x_1 = A_1^{-1}(b - A_2x_2) = A_1^{-1}b - A_1^{-1}A_2x_2.$$

This expression allows us to easily calculate a solution: we simply take $\hat{x}_2 = 0$, $\hat{x}_1 = A_1^{-1}b$. The cost is equal to the cost of solving one square set of p linear equations $A_1\hat{x}_1 = b$.

We can also parametrize all solutions of $Ax = b$, using $x_2 \in \mathbf{R}^{n-p}$ as a free parameter. The general solution of $Ax = b$ can be expressed as

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -A_1^{-1}A_2 \\ I \end{bmatrix} x_2 + \begin{bmatrix} A_1^{-1}b \\ 0 \end{bmatrix}.$$

This gives a parametrization of the form (C.12) with

$$F = \begin{bmatrix} -A_1^{-1}A_2 \\ I \end{bmatrix}, \qquad \hat{x} = \begin{bmatrix} A_1^{-1}b \\ 0 \end{bmatrix}.$$

To summarize, assume that the cost of factoring A_1 is f and the cost of solving one system of the form $A_1x = d$ is s. Then the cost of finding one solution of (C.11) is $f + s$. The cost of parametrizing all solutions (*i.e.*, calculating F and $\hat{x}$) is $f + s(n - p + 1)$.

Now we consider the general case, when the first p columns of A need not be independent. Since $\mathbf{rank}\, A = p$, we can select a set of p columns of A that is independent, permute them to the front, and then apply the method described

above. In other words, we find a permutation matrix P such that the first p columns of $\tilde{A} = AP$ are independent, *i.e.*,

$$\tilde{A} = AP = \begin{bmatrix} A_1 & A_2 \end{bmatrix},$$

where A_1 is invertible. The general solution of $\tilde{A}\tilde{x} = b$, where $\tilde{x} = P^T x$, is then given by

$$\tilde{x} = \begin{bmatrix} -A_1^{-1}A_2 \\ I \end{bmatrix} \tilde{x}_2 + \begin{bmatrix} A_1^{-1}b \\ 0 \end{bmatrix}.$$

The general solution of $Ax = b$ is then given by

$$x = P\tilde{x} = P \begin{bmatrix} -A_1^{-1}A_2 \\ I \end{bmatrix} z + P \begin{bmatrix} A_1^{-1}b \\ 0 \end{bmatrix},$$

where $z \in \mathbf{R}^{n-p}$ is a free parameter. This idea is useful when it is easy to identify a nonsingular or easily inverted submatrix of A, for example, a diagonal matrix with nonzero diagonal elements.

The QR factorization

If $C \in \mathbf{R}^{n\times p}$ with $p \leq n$ and $\mathbf{rank}\, C = p$, then it can be factored as

$$C = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where $Q_1 \in \mathbf{R}^{n\times p}$ and $Q_2 \in \mathbf{R}^{n\times(n-p)}$ satisfy

$$Q_1^T Q_1 = I, \qquad Q_2^T Q_2 = I, \qquad Q_1^T Q_2 = 0,$$

and $R \in \mathbf{R}^{p\times p}$ is upper triangular with nonzero diagonal elements. This is called the *QR factorization* of C. The QR factorization can be calculated in $2p^2(n - p/3)$ flops. (The matrix Q is stored in a factored form that makes it possible to efficiently compute matrix-vector products Qx and $Q^T x$.)

The QR factorization can be used to solve the underdetermined set of linear equations (C.11). Suppose

$$A^T = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}$$

is the QR factorization of A^T. Substituting in the equations it is clear that $\hat{x} = Q_1 R^{-T} b$ satisfies the equations:

$$A\hat{x} = R^T Q_1^T Q_1 R^{-T} b = b.$$

Moreover, the columns of Q_2 form a basis for the nullspace of A, so the complete solution set can be parametrized as

$$\{x = \hat{x} + Q_2 z \mid z \in \mathbf{R}^{n-p}\}.$$

The QR factorization method is the most common method for solving underdetermined equations. One drawback is that it is difficult to exploit sparsity. The factor Q is usually dense, even when C is very sparse.

LU factorization of a rectangular matrix

If $C \in \mathbf{R}^{n\times p}$ with $p \leq n$ and $\mathbf{rank}\, C = p$, then it can be factored as

$$C = PLU$$

where $P \in \mathbf{R}^{n\times n}$ is a permutation matrix, $L \in \mathbf{R}^{n\times p}$ is unit lower triangular (*i.e.*, $l_{ij} = 0$ for $i < j$ and $l_{ii} = 1$), and $U \in \mathbf{R}^{p\times p}$ is nonsingular and upper triangular. The cost is $(2/3)p^3 + p^2(n-p)$ flops if no structure in C is exploited.

If the matrix C is sparse, the LU factorization usually includes row and column permutations, *i.e.*, we factor C as

$$C = P_1 L U P_2$$

where $P_1,\ P_2 \in \mathbf{R}^{p\times p}$ are permutation matrices. The LU factorization of a sparse rectangular matrix can be calculated very efficiently, at a cost that is much lower than for dense matrices.

The LU factorization can be used to solve underdetermined sets of linear equations. Suppose $A^T = PLU$ is the LU factorization of the matrix A^T in (C.11), and we partition L as

$$L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix},$$

where $L_1 \in \mathbf{R}^{p\times p}$ and $L_2 \in \mathbf{R}^{(n-p)\times p}$. It is easily verified that the solution set can be parametrized as (C.12) with

$$\hat{x} = P \begin{bmatrix} L_1^{-T} U^{-T} b \\ 0 \end{bmatrix}, \qquad F = P \begin{bmatrix} -L_1^{-T} L_2^T \\ I \end{bmatrix}.$$