UCLA Fall Quarter 2011-12

Electrical Engineering 103
Applied Numerical Computing

Professor L. Vandenberghe

Notes written in collaboration with S. Boyd (Stanford Univ.)

Contents

Part I

Matrix theory

Chapter 1

Vectors

1.1 Definitions and notation

A *vector* is an ordered finite list of numbers. Vectors are usually written as vertical arrays, surrounded by brackets, as in

$$a = \begin{bmatrix} -1.1 \\ 0 \\ 3.6 \\ -7.2 \end{bmatrix}.$$

They can also be written as numbers separated by commas and surrounded by parentheses. In this notation style, the vector a defined above is written as

$$a = (-1.1, 0, 3.6, -7.2).$$

The *elements* (or *entries*, *coefficients*, *components*) of a vector are the values in the array. The *size* (also called *dimension* or *length*) of the vector is the number of elements it contains. The vector a above, for example, has size four. A vector of size n is called an *n-vector*. A 1-vector is considered to be the same as a number, *i.e.*, we do not distinguish between the 1-vector $[\ 1.3\ ]$ and the number 1.3.

The ith element of a vector a is denoted a_i, where i is an integer index that runs from 1 to n, the size of the vector. For the example above, a_3, the third element of the vector a, has the value 3.6.

The numbers or values of the elements in a vector are called *scalars*. We will focus on the case that arises in most applications, where the scalars are real numbers. In this case we refer to vectors as *real vectors*. The set of real n-vectors is denoted $\mathbf{R}^n$. Occasionally other types of scalars arise. For example, the scalars can be complex numbers, in which case we refer to the vector as a *complex vector*. The set of complex n-vectors is denoted $\mathbf{C}^n$. As another example, the scalars can be Boolean numbers.

Block vectors It is sometimes useful to define vectors by concatenating two or more vectors, as in

$$a = \begin{bmatrix} b \\ c \\ d \end{bmatrix}.$$

If b is an m-vector, c is an n-vector, and d is p-vector, this defines the $(m+n+p)$-vector

$$a = (b_1, b_2, \ldots, b_m, c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_p).$$

Notational conventions Some authors try to use notation that helps the reader distinguish between vectors and scalars (numbers). For example, Greek letters (α, β, ...) might be used for numbers, and lower-case letters (a, x, f, ...) for vectors. Other notational conventions include vectors given in bold font ($\mathbf{g}$), or vectors written with arrows above them ($\vec{a}$). These notational conventions are not standardized, so you should be prepared to figure out what things are (*i.e.*, scalars or vectors) despite the author's notational scheme (if any exists).

We should also give a couple of warnings concerning the subscripted index notation a_i. The first warning concerns the range of the index. In many computer languages, arrays are indexed from $i = 0$ to $n - 1$. But in standard mathematical notation, n-vectors are indexed from $i = 1$ to $i = n$, so in this book, vectors will be indexed from $i = 1$ to $i = n$. The next warning concerns an ambiguity in the notation a_i, used for the ith element of a vector a. The same notation will occasionally refer to the ith vector in a collection or list of k vectors a_1, ..., a_k. Whether a_3 means the third element of a vector a (in which case a_3 is a number), or the third vector in some list of vectors (in which case a_3 is a vector) will be clear from the context.

1.2 Zero and unit vectors

A zero vector is a vector with all elements equal to zero. Sometimes the zero vector of size n is written as 0_n, where the subscript denotes the size. But usually a zero vector is denoted just 0, the same symbol used to denote the number 0. In this case you have to figure out the size of the zero vector from the context. (We will see how this is done later.)

Even though zero vectors of different sizes are different vectors, we use the same symbol 0 to denote them. In programming this is called *overloading*: the symbol 0 is overloaded because it can mean different things depending on the context (*e.g.*, the equation it appears in).

A (standard) *unit vector* is a vector with all elements equal to zero, except one element which is equal to one. The ith unit vector (of size n) is the unit vector with ith element one, and is denoted e_i. The vectors

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

are the three unit vectors of size 3. The notation for unit vectors is an example of the ambiguity in notation noted at the end of section 1.1. Here, e_i denotes the ith unit vector, and not the ith element of a vector e. As with zero vectors, the size of e_i is usually determined from the context.

We use the notation $\mathbf{1}_n$ for the n-vector with all its elements equal to one. We also write $\mathbf{1}$ if the size of the vector can be determined from the context. (Some authors use e to denote a vector of all ones, but we will not use this notation.)

1.3 Vector addition

Two vectors *of the same size* can be added together by adding the corresponding elements, to form another vector of the same size. Vector addition is denoted by the symbol $+$. (Thus the symbol $+$ is overloaded to mean scalar addition when scalars appear on its left- and right-hand sides, and vector addition when vectors appear on its left- and right-hand sides.) For example,

$$\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 9 \\ 3 \end{bmatrix}.$$

Vector subtraction is similar. As an example,

$$\begin{bmatrix} 1 \\ 9 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \end{bmatrix}.$$

The following properties of vector addition are easily verified.

- Vector addition is commutative: if a and b are vectors of the same size, then $a + b = b + a$.
- Vector addition is associative: $(a + b) + c = a + (b + c)$. We can therefore write both as $a + b + c$.
- $a + 0 = 0 + a = a$. Adding the zero vector to a vector has no effect. (This is an example where the size of the zero vector follows from the context: it must be the same as the size of a.)

Some languages for manipulating vectors define (overload) the sum of a vector and a scalar as the vector obtained by adding the scalar to each element of the vector. This is not standard mathematical notation, however, so we will not use it. In our (more standard) notation, we can express the vector obtained by adding the scalar γ to each element of the vector a as $a + \gamma\mathbf{1}$ (using scalar multiplication, our next topic).

1.4 Scalar-vector multiplication

Another operation is *scalar multiplication* or *scalar-vector multiplication*, in which a vector is multiplied by a scalar (*i.e.*, number), which is done by multiplying every element of the vector by the scalar. Scalar multiplication is denoted by juxtaposition, with the scalar on the left, as in

$$(-2)\begin{bmatrix} 1 \\ 9 \\ 6 \end{bmatrix} = \begin{bmatrix} -2 \\ -18 \\ -12 \end{bmatrix}.$$

(Some people use scalar multiplication on the right, but this is nonstandard. Another nonstandard notation you might see is $a/2$ instead of $(1/2)a$.) The scalar product $(-1)a$ is written simply as $-a$. Note that $0\,a = 0$ (where the left-hand zero is the scalar zero, and the right-hand zero is a vector zero of the same size as a).

Scalar multiplication obeys several laws that are easy to figure out from the definition. If a is a vector and β, γ are scalars, then

$$(\beta + \gamma)a = \beta a + \gamma a.$$

Scalar multiplication, like ordinary multiplication, has higher precedence than vector addition, so the right-hand side here, $\beta a + \gamma a$, means $(\beta a) + (\gamma a)$. It is useful to identify the symbols appearing in this formula above. The $+$ symbol on the left is addition of scalars, while the $+$ symbol on the right denotes vector addition. Another simple property is $(\beta\gamma)a = \beta(\gamma a)$, where β and γ are scalars and a is a vector. On the left-hand side we see scalar-scalar multiplication $(\beta\gamma)$ and scalar-vector multiplication; on the right we see two scalar-vector products.

Linear combinations If $a_1, \ldots, a_m$ are n-vectors, and $\beta_1, \ldots, \beta_m$ are scalars, the n-vector

$$\beta_1 a_1 + \cdots + \beta_m a_m$$

is called a *linear combination* of the vectors $a_1, \ldots, a_n$. The scalars $\beta_1, \ldots, \beta_m$ are called the *coefficients* of the linear combination. As a simple but important application, we can write any vector a as a linear combination of the standard unit vectors, as

$$a = a_1 e_1 + \cdots + a_n e_n. \tag{1.1}$$

A specific example is

$$\begin{bmatrix} -1 \\ 3 \\ 5 \end{bmatrix} = (-1)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + 3\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + 5\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

1.5 Inner product

The (standard) *inner product* (also called *dot product*) of two n-vectors is defined as the scalar

$$a^T b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n.$$

(The origin of the superscript in a^T will be explained in chapter 2.) Some other notations for the inner product are $\langle a, b\rangle$, $\langle a|b\rangle$, (a, b), and $a \cdot b$. As you might guess, there is also a vector outer product, which we will encounter later, in chapter 2.

The inner product satisfies some simple properties that are easily verified from the definition. If a, b, and c are vectors of the same size, and γ is a scalar, we have the following.

- $a^T b = b^T a$. The inner product is commutative: the order of the two vector arguments does not matter.
- $(\gamma a)^T b = \gamma(a^T b)$, so we can write both as $\gamma a^T b$.
- $(a + b)^T c = a^T c + b^T c$. The inner product can be distributed across vector addition.

These can be combined to obtain other identities, such as $a^T(\gamma b) = \gamma(a^T b)$, or $a^T(b + \gamma c) = a^T b + \gamma a^T c$.

Examples

- $e_i^T a = a_i$. The inner product of a vector with the ith standard unit vector gives (or 'picks out') the ith element a.
- $\mathbf{1}^T a = a_1 + \cdots + a_n$. The inner product of a vector with the vector of ones gives the sum of the elements of the vector.
- $a^T a = a_1^2 + \cdots + a_n^2$. The inner product of a vector with itself gives the sum of the squares of the elements of the vector.
- If a and b are n-vectors, each of whose elements are either 0 or 1, then $a^T b$ gives the total number of indices for which a_i and b_i are both one.

1.6 Linear functions

The notation $f : \mathbf{R}^n \to \mathbf{R}$ means that f is a function that maps real n-vectors to real numbers, *i.e.*, it is a scalar valued function of n-vectors. If x is an n-vector, then $f(x)$ denotes the value of the function f at x, which is a scalar. We can also interpret f as a function of n scalars, in which case we write $f(x)$ as

$$f(x) = f(x_1, x_2, \ldots, x_n).$$

1.6.1 Inner products and linear functions

Suppose a is an n-vector. We can define a scalar valued function f of n-vectors, given by

$$f(x) = a^T x = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \tag{1.2}$$

for any n-vector x. This function gives the inner product of its n-dimensional argument x with some (fixed) n-vector a. We can also think of f as forming a weighted sum of the elements of x; the elements of a give the weights.

Superposition and linearity The function f defined in (1.2) satisfies the property

$$\begin{aligned} f(\alpha x + \beta y) &= a^T(\alpha x + \beta y) \\ &= a^T(\alpha x) + a^T(\beta y) \\ &= \alpha(a^T x) + \beta(a^T y) \\ &= \alpha f(x) + \beta f(y) \end{aligned}$$

for all n-vectors x, y, and all scalars α, β. This property is called *superposition*. A function that satisfies the superposition property is called *linear*. We have just showed that the inner product with a fixed vector is a linear function.

The superposition equality

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{1.3}$$

looks deceptively simple; it is easy to read it as just a re-arrangement of the parentheses and the order of a few terms. But in fact it says a lot. On the left-hand side, the term $\alpha x + \beta y$ involves *vector addition* and *scalar-vector* multiplication. On the right-hand side, $\alpha f(x) + \beta f(y)$ involves ordinary *scalar multiplication* and *scalar addition*.

Inner product representation of a linear function The converse is also true: If a function is linear, then it can be expressed as the inner product with some fixed vector. Suppose f is a scalar valued function of n-vectors, and is linear, *i.e.*,

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

for all n-vectors x, y, and all scalars α, β. Then there is an n-vector a such that $f(x) = a^T x$ for all x. We call $a^T x$ the inner product representation of f.

To see this, we use the identity (1.1) to express an arbitrary n-vector x as $x = x_1 e_1 + \cdots + x_n e_n$. If f is linear, then

$$\begin{aligned} f(x) &= f(x_1 e_1 + x_2 e_2 + \cdots + x_n e_n) \\ &= x_1 f(e_1) + f(x_2 e_2 + \cdots + x_n e_n) \\ &= x_1 f(e_1) + x_2 f(e_2) + f(x_3 e_3 + \cdots + x_n e_n) \\ &\vdots \\ &= x_1 f(e_1) + x_2 f(e_2) + \cdots + x_n f(e_n) \\ &= a^T x \end{aligned}$$

with $a = (f(e_1), f(e_2), \ldots, f(e_n))$. The formula just derived,

$$f(x) = x_1 f(e_1) + x_2 f(e_2) + \cdots + x_n f(e_n) \tag{1.4}$$

which holds for any linear scalar valued function f, has several interesting implications. Suppose, for example, that the linear function f is given as a subroutine (or a physical system) that computes (or results in the output) $f(x)$ when we give the argument (or input) x. Once we have found $f(e_1), \ldots, f(e_n)$, by n calls to the subroutine (or n experiments), we can predict (or simulate) what $f(x)$ will be, for *any* vector x, using the formula (1.4).

The representation of a linear function f as $f(x) = a^T x$ is unique. In other words, there is only one vector a for which $f(x) = a^T x$ holds for all x. To see this, suppose that we have $f(x) = a^T x$ for all x, and also $f(x) = b^T x$ for all x. Taking $x = e_i$, we have $f(e_i) = a^T e_i = a_i$, using the formula $f(x) = a^T x$. Using the formula $f(x) = b^T x$, we have $f(e_i) = b^T e_i = b_i$. These two numbers must be the same, so we have $a_i = b_i$. Repeating this argument for $i = 1, \ldots, n$, we conclude that the corresponding elements in a and b are the same, so $a = b$.

Examples

- *Average.* The average of the elements of x, $f(x) = (x_1 + x_2 + \cdots + x_n)/n$, is a linear function. It can be expressed as $f(x) = a^T x$ with

$$a = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}) = \frac{1}{n}\mathbf{1}.$$

- *Maximum.* The maximum element of x, $f(x) = \max_k x_k$, is not a linear function. We can show this is by a counterexample. Choose $n = 2$, $x = (1, -1)$, $y = (-1, 1)$, $\alpha = 1$, $\beta = 1$. Then

$$f(\alpha x + \beta y) = 0 \neq \alpha f(x) + \beta f(y) = 2.$$

Affine functions A linear function plus a constant is called an *affine* function. A function $f : \mathbf{R}^n \to \mathbf{R}$ is affine if and only if it can be expressed as $f(x) = a^T x + b$ for some n-vector a and scalar b. For example, the function on 3-vectors defined by

$$f(x) = 2.3 - 2x_1 + 1.3x_2 - x_3,$$

is affine, with $b = 2.3$, $a = (-2, 1.3, -1)$.

Any affine scalar valued function satisfies the following variation on the superposition property:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y),$$

for all n-vectors x, y, and all scalars α, β that satisfy $\alpha + \beta = 1$. (In the definition of linearity, there is no requirement that the coefficients α and β sum to one.) The converse is also true: Any scalar valued function that satisfies this property is affine. An analog of the formula (1.4) for linear functions is

$$f(x) = f(0) + x_1 \left(f(e_1) - f(0)\right) + \cdots + x_n \left(f(e_n) - f(0)\right),$$

which holds when f is affine, and x is any n-vector (see exercise 1.19). This formula shows that for an affine function, once we know the $n+1$ numbers $f(0)$, $f(e_1)$, $\ldots$, $f(e_n)$, we can predict (or reconstruct or evaluate) $f(x)$ for any n-vector x.

In some contexts affine functions are called linear. For example, $y = \alpha x + \beta$ is sometimes referred to as a linear function of x. As another common example, the first-order Taylor approximation of f (described below) is in general an affine function, but is often called linear. In this book, however, we will distinguish between linear and affine functions.

1.6.2 Linearization of a scalar valued function

Suppose $f : \mathbf{R}^n \to \mathbf{R}$ is continuously differentiable, *i.e.*, the partial derivatives

$$\frac{\partial f(\hat{x})}{\partial x_k} = \lim_{t \to 0} \frac{f(\hat{x} + te_k) - f(\hat{x})}{t}, \quad k = 1, \ldots, n,$$

exist at all $\hat{x}$, and are continuous functions of $\hat{x}$. The *gradient* of f, at a vector $\hat{x}$, is the vector of first partial derivatives evaluated at $\hat{x}$, and is denoted $\nabla f(\hat{x})$:

$$\nabla f(\hat{x}) = \begin{bmatrix} \partial f(\hat{x})/\partial x_1 \\ \partial f(\hat{x})/\partial x_2 \\ \vdots \\ \partial f(\hat{x})/\partial x_n \end{bmatrix}.$$

The *first-order Taylor approximation* of f around (or near) $\hat{x}$ is defined as the function f_{aff} defined by

$$\begin{aligned} f_{\text{aff}}(x) &= f(\hat{x}) + \sum_{k=1}^{n} \frac{\partial f(\hat{x})}{\partial x_k}(x_k - \hat{x}_k) \\ &= f(\hat{x}) + \nabla f(\hat{x})^T (x - \hat{x}). \end{aligned} \tag{1.5}$$

The function f_{aff} is also called the *linearization* of f at (or near) the point $\hat{x}$, and gives a very good approximation of $f(x)$ for x near $\hat{x}$. If $n = 1$, the gradient is simply the first derivative $f'(\hat{x})$, and the first-order approximation reduces to

$$f_{\text{aff}}(x) = f(\hat{x}) + f'(\hat{x})(x - \hat{x}).$$

The first-order Taylor approximation f_{aff} is an affine function of x; we can write it as $f_{\text{aff}}(x) = a^T x + b$, where

$$a = \nabla f(\hat{x}), \qquad b = f(\hat{x}) - \nabla f(\hat{x})^T \hat{x}$$

(but the formula given in (1.5) is probably easier to understand).

The first-order approximation is often expressed in the following way. Define $y = f(x)$ and $\hat{y} = f(\hat{x})$. We define δy and δx (which are to be interpreted as indivisible two-character symbols, not the products of δ and y or x) as

$$\delta y = y - \hat{y}, \qquad \delta x = x - \hat{x}.$$

These are called the y and x *deviations* (from $\hat{y}$ and $\hat{x}$) respectively. The first-order approximation can be expressed as

$$\delta y \approx \nabla f(\hat{x})^T \delta x,$$

for x near $\hat{x}$. Thus, the y deviation is approximately a linear function of the x deviation.

Example We will find the first-order Taylor approximation of the function $f : \mathbf{R}^2 \to \mathbf{R}$, defined by

$$f(x) = e^{x_1+x_2-1} + e^{x_1-x_2-1} + e^{-x_1-1}, \tag{1.6}$$

at the point $\hat{x} = 0$. Its gradient, at a general point $\hat{x}$, is

$$\nabla f(\hat{x}) = \begin{bmatrix} e^{\hat{x}_1+\hat{x}_2-1} + e^{\hat{x}_1-\hat{x}_2-1} - e^{-\hat{x}_1-1} \\ e^{\hat{x}_1+\hat{x}_2-1} - e^{\hat{x}_1-\hat{x}_2-1} \end{bmatrix}, \tag{1.7}$$

so $\nabla f(0) = (1/e, 0)$. The first-order approximation of f around $\hat{x} = 0$ is

$$\begin{aligned} f_{\text{aff}}(x) &= f(0) + \nabla f(0)^T (x - 0) \\ &= 3/e + (1/e, 0)^T x \\ &= 3/e + x_1/e. \end{aligned}$$

For x_1 and x_2 near zero, the function $(x_1 + 3)/e$ therefore gives a very good approximation of $f(x)$.

1.7 Euclidean norm

The *Euclidean norm* of a vector x, denoted $\|x\|$, is the square root of the sum of the squares if its elements,

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

The Euclidean norm is sometimes written with a subscript 2, as $\|x\|_2$. We can express the Euclidean norm in terms of the inner product of x with itself:

$$\|x\| = \sqrt{x^T x}.$$

When x is a scalar, *i.e.*, a 1-vector, the Euclidean norm is the same as the absolute value of x. Indeed, the Euclidean norm can be considered a generalization or extension of the absolute value or magnitude, that applies to vectors. The double bar notation is meant to suggest this.

Some important properties of norms are given below. Here x and y are vectors, and β is a scalar.

- *Homogeneity.* $\|\beta x\| = |\beta|\|x\|$.
- *Triangle inequality.* $\|x + y\| \leq \|x\| + \|y\|$.
- *Nonnegativity.* $\|x\| \geq 0$.
- *Definiteness.* $\|x\| = 0$ only if $x = 0$.

The last two properties together, which state that the norm is always nonnegative, and zero only when the vector is zero, are called *positive definiteness*. The first, third, and fourth properties are easy to show directly from the definition of the norm. Establishing the second property, the triangle inequality, is not as easy; we will give a derivation a bit later.

Root-mean-square value The Euclidean norm is related to the *root-mean-square* (RMS) value of a vector a, defined as

$$\text{RMS}(x) = \sqrt{\frac{1}{n}\left(x_1^2 + \cdots + x_n^2\right)} = \frac{1}{\sqrt{n}}\|x\|.$$

Roughly speaking, the RMS value of a vector x tells us what a 'typical' value of $|x_i|$ is, and is useful when comparing norms of vectors of different sizes.

Euclidean distance We can use the norm to define the *Euclidean distance* between two vectors a and b as the norm of their difference:

$$\mathbf{dist}(a, b) = \|a - b\|.$$

For dimensions one, two, and three, this distance is exactly the usual distance between points with coordinates a and b. But the Euclidean distance is defined for vectors of any dimension; we can refer to the distance between two vectors of dimension 100.

We can now explain where the triangle inequality gets its name. Consider a triangle in dimension two or three, whose vertices have coordinates a, b, and c. The lengths of the sides are the distances between the vertices,

$$\|a - b\|, \qquad \|b - c\|, \qquad \|a - c\|.$$

The length of any side of a triangle cannot exceed the sum of the lengths of the other two sides. For example, we have

$$\|a - c\| \leq \|a - b\| + \|b - c\|. \tag{1.8}$$

This follows from the triangle inequality, since

$$\|a - c\| = \|(a - b) + (b - c)\| \leq \|a - b\| + \|b - c\|.$$

General norms Any real valued function f that satisfies the four properties above (homogeneity, triangle inequality, nonnegativity, and definiteness) is called a *vector norm*, and is usually written as $f(x) = \|x\|_{\mathrm{mn}}$, where the subscript is some kind of identifier or mnemonic to identify it.

Two common vector norms are the 1-norm $\|a\|_1$ and the ∞-norm $\|a\|_\infty$, which are defined as

$$\|a\|_1 = |a_1| + |a_2| + \cdots + |a_n|, \qquad \|a\|_\infty = \max_{k=1,\ldots,n} |a_k|.$$

These norms measure the sum and the maximum of the absolute values of the elements in a vector, respectively. The 1-norm and the ∞-norm arise in some recent and advanced applications, but we will not encounter them much in this book. When we refer to the norm of a vector, we always mean the Euclidean norm.

Weighted norm Another important example of a general norm is a *weighted norm*, defined as

$$\|x\|_w = \sqrt{(x_1/w_1)^2 + \cdots + (x_n/w_n)^2},$$

where $w_1, \ldots, w_n$ are given positive *weights*, used to assign more or less importance to the different elements of the n-vector x. If all the weights are one, the weighted norm reduces to the ('unweighted') Euclidean norm.

Weighted norms arise naturally when the elements of the vector x have different physical units, or natural ranges of values. One common rule of thumb is to choose w_i equal to the typical value of $|x_i|$ in the application or setting. These weights bring all the terms in the sum to the same order, one. We can also imagine that the weights contain the same physical units as the elements x_i, which makes the terms in the sum (and therefore the norm as well) unitless.

For example, consider an application in which x_1 represents a physical distance, measured in meters, and has typical values on the order of $\pm 10^5$m, and x_2 represents a physical velocity, measured in meters/sec, and has typical values on the order of 10m/s. In this case we might wish to use the weighted norm

$$\|x\|_w = \sqrt{(x_1/w_1)^2 + (x_2/w_2)^2},$$

with $w_1 = 10^5$ and $w_2 = 10$, to measure the size of a vector. If we assign to w_1 physical units of meters, and to w_2 physical units of meters per second, the weighted norm $\|x\|_w$ becomes unitless.

1.8 Angle between vectors

Cauchy-Schwarz inequality An important inequality that relates Euclidean norms and inner products is the *Cauchy-Schwarz inequality:*

$$|a^T b| \leq \|a\| \, \|b\|$$

for all a and b. This can be shown as follows. The inequality clearly holds if $a = 0$ or $b = 0$ (in this case, both sides of the inequality are zero). Suppose $a \neq 0$, $b \neq 0$,

and consider the function $f(t) = \|a + tb\|^2$, where t is a scalar. We have $f(t) \geq 0$ for all t, since $f(t)$ is a sum of squares. By expanding the square of the norm we can write f as

$$\begin{aligned} f(t) &= (a + tb)^T(a + tb) \\ &= a^Ta + tb^Ta + ta^Tb + t^2b^Tb \\ &= \|a\|^2 + 2ta^Tb + t^2\|b\|^2. \end{aligned}$$

This is a quadratic function with a positive coefficient of t^2. It reaches its minimum where $f'(t) = 2a^Tb + 2t\|b\|^2 = 0$, *i.e.*, at $\bar{t} = -a^Tb/\|b\|^2$. The value of f at the minimum is

$$f(\bar{t}) = \|a\|^2 - \frac{(a^Tb)^2}{\|b\|^2}.$$

This must be nonnegative since $f(t) \geq 0$ for all t. Therefore $(a^Tb)^2 \leq \|a\|^2\|b\|^2$. Taking the square root on each side gives the Cauchy-Schwarz inequality.

This argument also reveals the conditions on a and b under which they satisfy the Cauchy-Schwarz inequality with equality. Suppose a and b are nonzero, with $|a^Tb| = \|a\|\,\|b\|$. Then

$$f(\bar{t}) = \|a + \bar{t}b\|^2 = \|a\|^2 - (a^Tb)^2/\|b\|^2 = 0,$$

and so $a + \bar{t}b = 0$. Thus, if the Cauchy-Schwarz inequality holds with equality for nonzero a and b, then a and b are scalar multiples of each other. If $\bar{t} < 0$, a is a positive multiple of b (and we say the vectors are *aligned*). In this case we have

$$a^Tb = -\bar{t}b^Tb = |\bar{t}|\|b\|^2 = \|-\bar{t}b\|\,\|b\| = \|a\|\,\|b\|.$$

If $\bar{t} > 0$, a is a negative multiple of b (and we say the vectors are *anti-aligned*). We have

$$a^Tb = -\bar{t}b^Tb = -|\bar{t}|\|b\|^2 = -\|-\bar{t}b\|\|b\| = -\|a\|\|b\|.$$

Verification of triangle inequality We can use the Cauchy-Schwarz inequality to verify the triangle inequality. Let a and b be any vectors. Then

$$\begin{aligned} \|a + b\|^2 &= \|a\|^2 + 2a^Tb + \|b\|^2 \\ &\leq \|a\|^2 + 2\|a\|\|b\| + \|b\|^2 \\ &= \left(\|a\| + \|b\|\right)^2, \end{aligned}$$

where we used the Cauchy-Schwarz inequality in the second line. Taking the square root we get the triangle inequality, $\|a + b\| \leq \|a\| + \|b\|$.

Correlation coefficient Suppose a and b are nonzero vectors of the same size. We define their *correlation coefficient* as

$$\rho = \frac{a^Tb}{\|a\|\,\|b\|}.$$

This is a symmetric function of the vectors: the correlation between a and b is the same as the correlation coefficient between b and a. The Cauchy-Schwarz

inequality tells us that the correlation coefficient ranges between -1 and $+1$. For this reason, the correlation coefficient is sometimes expressed as a percentage. For example, $\rho = 30\%$ means $\rho = 0.3$, *i.e.*, $a^T b = 0.3\|a\|\|b\|$. Very roughly speaking, the correlation coefficient ρ tells us how well the shape of one vector (say, if it were plotted versus index) matches the shape of the other. We have already seen, for example, that $\rho = 1$ only if the vectors are aligned, which means that each is a positive multiple of the other, and that $\rho = -1$ occurs only when each vector is a negative multiple of the other.

Two vectors are said to be *highly correlated* if their correlation coefficient is near one, and *uncorrelated* if the correlation coefficient is small.

Example Conisider

$$x = \begin{bmatrix} 0.1 \\ -0.3 \\ 1.3 \\ -0.3 \\ -3.3 \end{bmatrix} \qquad y = \begin{bmatrix} 0.2 \\ -0.4 \\ 3.2 \\ -0.8 \\ -5.2 \end{bmatrix} \qquad z = \begin{bmatrix} 1.8 \\ -1.0 \\ -0.6 \\ 1.4 \\ -0.2 \end{bmatrix}.$$

We have

$$\|x\| = 3.57, \qquad \|y\| = 6.17, \qquad \|z\| = 2.57, \qquad x^T y = 21.70, \qquad x^T z = -0.06,$$

and therefore

$$\rho_{xy} = \frac{x^T y}{\|x\|\,\|y\|} = 0.98, \qquad \rho_{xz} = \frac{x^T z}{\|x\|\,\|z\|} = -0.007.$$

We see that x and y are highly correlated (y is roughly equal to x scaled by 2). The vectors x and z on the other hand are almost uncorrelated.

Angle between vectors The *angle* between two nonzero vectors a, b with correlation coefficient ρ is defined as

$$\theta = \arccos\rho = \arccos\left(\frac{a^T b}{\|a\|\,\|b\|}\right)$$

where arccos denotes the inverse cosine, normalized to lie in the interval $[0, \pi]$. In other words, we define θ as the unique number in $[0, \pi]$ that satisfies

$$a^T b = \|a\|\,\|b\|\cos\theta.$$

The angle between a and b is sometimes written as $\angle(a, b)$. The angle is a symmetric function of a and b: we have $\angle(a, b) = \angle(b, a)$. The angle between vectors is sometimes expressed in degrees. For example, $\angle(a, b) = 30^\circ$ means $\angle(a, b) = \pi/6$, *i.e.*, $a^T b = (1/2)\|a\|\|b\|$.

The angle coincides with the usual notion of angle between vectors, when they have dimension two or three. But the definition of angle is more general; we can refer to the angle between two vectors with dimension 100.

Acute and obtuse angles Angles are classified according to the sign of $a^T b$.

- If the angle is $\pi/2 = 90^\circ$, *i.e.*, $a^T b = 0$, the vectors are said to be *orthogonal*. This is the same as the vectors being uncorrelated. We write $a \perp b$ if a and b are orthogonal.
- If the angle is zero, which means $a^T b = \|a\|\|b\|$, the vectors are *aligned*. This is the same as saying the vectors have correlation coefficient 1, or that each vector is a positive multiple of the other (assuming the vectors are nonzero).
- If the angle is $\pi = 180^\circ$, which means $a^T b = -\|a\|\,\|b\|$, the vectors are *anti-aligned*. This means the correlation coefficient is -1, and each vector is a negative multiple of the other (assuming the vectors are nonzero).
- If $\angle(a, b) \leq \pi/2 = 90^\circ$, the vectors are said to make an *acute angle*. This is the same as saying the vectors have nonnegative correlation coefficient, or nonnegative inner product, $a^T b \geq 0$.
- If $\angle(a, b) \geq \pi/2 = 90^\circ$, the vectors are said to make an *obtuse angle*. This is the same as saying the vectors have nonpositive correlation coefficient, or nonpositive inner product, $a^T b \leq 0$.

Orthonormal set of vectors A vector a is said the *normalized*, or a *unit vector*, if $\|a\| = 1$. A set of vectors $\{a_1, \ldots, a_k\}$ is *orthogonal* if $a_i \perp a_j$ for any i, j with $i \neq j$, $i, j = 1, \ldots, k$. It is important to understand that orthogonality is an attribute of a *set* of vectors, and not an attribute of vectors individually. It makes no sense to say that a vector a is orthogonal; but we can say that $\{a\}$ (the set whose only element is a) is orthogonal. But this last statement is true for any nonzero a, and so is not informative.

A set of normalized and orthogonal vectors is also called *orthonormal*. For example, the set of 3-vectors

$$\left\{ \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \right\}$$

is orthonormal.

Orthonormal sets of vectors have many applications. Suppose a vector x is a linear combination of $a_1, \ldots, a_k$, where $\{a_1, \ldots, a_k\}$ is an orthonormal set of vectors,

$$x = \beta_1 a_1 + \cdots + \beta_k a_k.$$

Taking the inner product of the left and right-hand sides of this equation with a_i yields

$$\begin{aligned} a_i^T x &= a_i^T (\beta_1 a_1 + \cdots + \beta_k a_k) \\ &= \beta_1 a_i^T a_1 + \cdots + \beta_k a_i^T a_k \\ &= \beta_i, \end{aligned}$$

since $a_i^T a_j = 1$ for $i = j$, and $a_i^T a_j = 0$ for $i \neq j$. So if a vector can be expressed as a linear combination of an orthonormal set of vectors, we can easily find the coefficients of the linear combination by taking the inner products with the vectors.

1.9 Vector inequality

In some applications it is useful to define inequality between vectors. If x and y are vectors of the same size, say n, we define $x \leq y$ to mean that $x_1 \leq y_1$, ..., $x_n \leq y_n$, *i.e.*, each element of x is less than or equal to the corresponding element of y. We define $x \geq y$ in the obvious way, to mean $x_1 \geq y_1$, ..., $x_n \geq y_n$. We refer to $x \leq y$ or $x \geq y$ as (nonstrict) *vector inequalities*.

We define strict inequalities in a similar way: $x < y$ means $x_1 < y_1$, ..., $x_n < y_n$, and $x > y$ means $x_1 > y_1$, ..., $x_n > y_n$. these are called *strict vector inequalities*.

Exercises

Block vectors

1.1 *Some block vector operations.* Let x be a block vector with two vector elements,

$$x = \begin{bmatrix} a \\ b \end{bmatrix},$$

where a and b are vectors of size n and m, respectively.

(a) Show that

$$\gamma x = \begin{bmatrix} \gamma a \\ \gamma b \end{bmatrix},$$

where γ is a scalar.

(b) Show that

$$\|x\| = \left(\|a\|^2 + \|b\|^2\right)^{1/2} = \left\| \begin{bmatrix} \|a\| \\ \|b\| \end{bmatrix} \right\|.$$

(Note that the norm on the right-hand side is of a 2-vector.)

(c) Let y be another block vector

$$y = \begin{bmatrix} c \\ d \end{bmatrix},$$

where c and d are vectors of size n and m, respectively. Show that

$$x + y = \begin{bmatrix} a + c \\ b + d \end{bmatrix}, \qquad x^T y = a^T c + b^T d.$$

Linear functions

1.2 Which of the following scalar valued functions on $\mathbf{R}^n$ are linear? Which are affine? If a function is linear, give its inner product representation, *i.e.*, an n-vector a such that $f(x) = a^T x$ for all x. If it is affine, give a and b such that $f(x) = a^T x + b$ holds for all x. If it is neither, give specific x, y, α, and β for which superposition fails, *i.e.*,

$$f(\alpha x + \beta y) \neq \alpha f(x) + \beta f(y).$$

(Provided $\alpha + \beta = 1$, this shows the function is neither linear nor affine.)

(a) The spread of values of the vector, defined as $f(x) = \max_k x_k - \min_k x_k$.

(b) The difference of the last element and the first, $f(x) = x_n - x_1$.

(c) The difference of the squared distances to two fixed vectors c and d, defined as

$$f(x) = \|x - c\|^2 - \|x - d\|^2.$$

1.3 A unit mass moves on a straight line (in one dimension). The position of the mass at time t is denoted by $s(t)$, and its derivatives (the velocity and acceleration) by $s'(t)$ and $s''(t)$. The position as a function of time can be determined from Newton's second law

$$s''(t) = F(t),$$

where $F(t)$ is the force applied at time t, and the initial conditions $s(0)$, $s'(0)$. We assume $F(t)$ is piecewise-constant, and is kept constant in intervals of one second. The sequence of forces $F(t)$, for $0 \leq t < 10$s, can then be represented by a 10-vector x, with

$$F(t) = x_k, \quad k - 1 \leq t < k.$$

(a) Suppose the initial position and velocity are zero ($s(0) = s'(0) = 0$). Derive expressions for the velocity $s'(10)$ and position $s(10)$ at time $t = 10$. Show that $s(10)$ and $s'(10)$ are linear functions of x.

(b) How does the answer change if we start from nonzero initial position or velocity?

1.4 *Deviation of middle element value from average.* Suppose x is a n-vector, with $n = 2m-1$ and $m \geq 1$. We define the middle element value of x as x_m. Define

$$f(x) = x_m - \frac{1}{n}\sum_{i=1}^{n} x_i,$$

which is the difference between the middle element value and the average of the coefficients in x. Express f in the form $f(x) = a^T x$, where a is an n-vector.

1.5 The temperature T of an electronic device containing three processors is an affine function of the power dissipated by the three processors, $P = (P_1, P_2, P_3)$. When all three processors are idling, we have $P = (10, 10, 10)$, which results in a temperature $T = 30$. When the first processor operates at full power and the other two are idling, we have $P = (100, 10, 10)$, and the temperature rises to $T = 60$. When the second processor operates at full power and the other two are idling, we have $P = (10, 100, 10)$ and $T = 70$. When the third processor operates at full power and the other two are idling, we have $P = (10, 10, 100)$ and $T = 65$. Now suppose that all three processors are operated at the same power P. How large can P be, if we require that $T \leq 85$?

1.6 *Taylor approximation of norm.* Find an affine approximation of the function $f(x) = \|x\|$, near a given nonzero vector $\hat{x}$. Check your approximation formula as follows. First, choose $\hat{x}$ as a random 5-vector. (In MATLAB, for example, you can use the command `hatx = randn(5,1)`.) Then choose several vectors x randomly, with $\|x - \hat{x}\|$ on the order of $0.1\|\hat{x}\|$ (in MATLAB, using `x = hatx + 0.1*randn(5,1)`), and compare $f(x) = \|x\|$ and $f(\hat{x}) = \|\hat{x}\|$.

Norms and angles

1.7 Verify that the following identities hold for any two vectors a and b of the same size.

(a) $(a+b)^T(a-b) = \|a\|^2 - \|b\|^2$.

(b) $\|a+b\|^2 + \|a-b\|^2 = 2(\|a\|^2 + \|b\|^2)$.

1.8 When does the triangle inequality hold with equality, *i.e.*, what are the conditions on a and b to have $\|a+b\| = \|a\| + \|b\|$?

1.9 Show that $\|a+b\| \geq \big|\|a\| - \|b\|\big|$.

1.10 *Approximating one vector with a scalar multiple of another.* Suppose we have a nonzero vector x and a vector y of the same size.

(a) How do you choose the scalar t so that $\|tx - y\|$ is minimized? Since $L = \{tx \mid t \in \mathbf{R}\}$ describes the line passing through the origin and the point x, the problem is to find the point in L closest to the point y. *Hint.* Work with $\|tx - y\|^2$.

(b) Let $t^\star$ be the value found in part (a), and let $z = t^\star x$. Show that $(y - z) \perp x$.

(c) Draw a picture illustrating this exercise, for 2-vectors. Show the points x and y, the line L, the point z, and the line segment between y and z.

(d) Express $d = \|z - y\|$, which is the closest distance of a point on the line L and the point y, in terms of $\|y\|$ and $\theta = \angle(x, y)$.

1.11 *Minimum distance between two lines.* Find the minimum distance between the lines

$$L = \{u + tv \mid t \in \mathbf{R}\}, \qquad \tilde{L} = \{\tilde{u} + \tilde{t}\tilde{v} \mid \tilde{t} \in \mathbf{R}\}.$$

The vectors u, v, $\tilde{u}$, and $\tilde{v}$ are given n-vectors with $v \neq 0$, $\tilde{v} \neq 0$.

1.12 *Hölder inequality.* The Cauchy-Schwarz inequality states that $|x^T y| \leq \|x\|\|y\|$, for any vectors x and y. The Hölder inequality gives another bound on the absolute value of the inner product, in terms of the product of norms: $|x^T y| \leq \|x\|_\infty \|y\|_1$. (These are the ∞ and 1-norms, defined on page 13.) Derive the Hölder inequality.

1.13 *Norm of linear combination of orthonormal vectors.* Suppose $\{a_1, \ldots, a_k\}$ is an orthonormal set of n-vectors, and $x = \beta_1 a_1 + \cdots + \beta_k a_k$. Express $\|x\|$ in terms of β_1, $\ldots$, β_k.

1.14 *Relative deviation between vectors.* Suppose a and b are nonzero vectors of the same size. The relative deviation of b from a is defined as the distance between a and b, divided by the norm of a,

$$\eta_{ab} = \frac{\|a - b\|}{\|a\|}.$$

This is often expressed as a percentage. The relative deviation is not a symmetric function of a and b; in general, $\eta_{ab} \neq \eta_{ba}$.

Suppose $\eta_{ab} = 0.1$ (*i.e.*, 10%). How big and how small can be η_{ba} be? Explain your reasoning.

1.15 *Average and norm.* Use the Cauchy-Schwarz inequality to prove that

$$-\frac{1}{\sqrt{n}}\|x\| \leq \frac{1}{n}\sum_{i=1}^{n} x_i \leq \frac{1}{\sqrt{n}}\|x\|$$

for all n-vectors x. In other words, the average of the elements of a vector lies between $\pm 1/\sqrt{n}$ times its norm. What are the conditions on x to have equality in the upper bound? When do we have equality in the lower bound?

1.16 Use the Cauchy-Schwarz inequality to prove that

$$\frac{1}{n}\sum_{k=1}^{n} x_k \geq \left(\frac{1}{n}\sum_{k=1}^{n} \frac{1}{x_k}\right)^{-1}$$

for all n-vectors x with positive elements x_k.

The left-hand side of the inequality is the arithmetic mean (average) of the numbers x_k; the right-hand side is called the harmonic mean.

1.17 *Euclidean norm of sum.* Derive a formula for $\|a+b\|$ in terms of $\|a\|$, $\|b\|$, and $\theta = \angle(a, b)$. Use this formula to show the following:

(a) $a \perp b$ if and only if $\|a + b\| = \sqrt{\|a\|^2 + \|b\|^2}$.

(b) a and b make an acute angle if and only if $\|a + b\| \geq \sqrt{\|a\|^2 + \|b\|^2}$.

(c) a and b make an obtuse angle if and only if $\|a + b\| \leq \sqrt{\|a\|^2 + \|b\|^2}$.

Draw a picture illustrating each case (in $\mathbf{R}^2$).

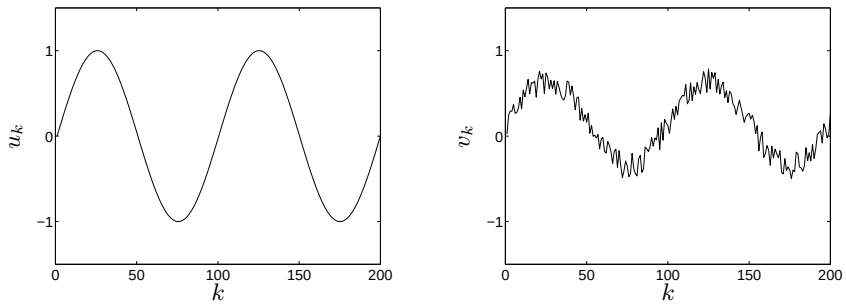
1.18 *Angle between nonnegative vectors.* Show that the angle between two nonnegative vectors x and y lies between 0 and $\pi/2$. When are two nonnegative vectors x and y orthogonal?

1.19 Let $f : \mathbf{R}^n \to \mathbf{R}$ be an affine function. Show that f can be expressed as $f(x) = a^T x + b$ for some $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$.

1.20 *Decoding using inner products.* An input signal is sent over a noisy communication channel. The channel adds a small noise to the input signal, and attenuates it by an unknown factor α. We represent the input signal as an n-vector u, the output signal as an n-vector v, and the noise signal as an n-vector w. The elements u_k, v_k, w_k give the values of the signals at time k. The relation between u and v is

$$v = \alpha u + w.$$

The two plots below show an example with $\alpha = 0.5$.



Now suppose we know that the input signal u was chosen from a set of four possible signals

$$x^{(1)},\ x^{(2)},\ x^{(3)},\ x^{(4)} \in \mathbf{R}^n.$$

We know these signals $x^{(i)}$, but we do not know which one was used as input signal u. Our task is to find a simple, automated way to estimate the input signal, based on the received signal v. There are many ways to do this. One possibility is to calculate the angle θ_k between v and $x^{(k)}$, via the formula

$$\cos\theta_k = \frac{v^T x^{(k)}}{\|v\|\,\|x^{(k)}\|}$$

and pick the signal $x^{(k)}$ that makes the smallest angle with v.

Download the file `ch1ex20.m` from the class webpage, save it in your working directory, and execute it in MATLAB using the command `[x1,x2,x3,x4,v] = ch1ex20`. The first four output arguments are the possible input signals $x^{(k)}$, $k = 1, 2, 3, 4$. The fifth output argument is the received (output) signal v. The length of the signals is $n = 200$.

(a) Plot the vectors v, $x^{(1)}$, $x^{(2)}$, $x^{(3)}$, $x^{(4)}$. Visually, it should be obvious which input signal was used to generate v.

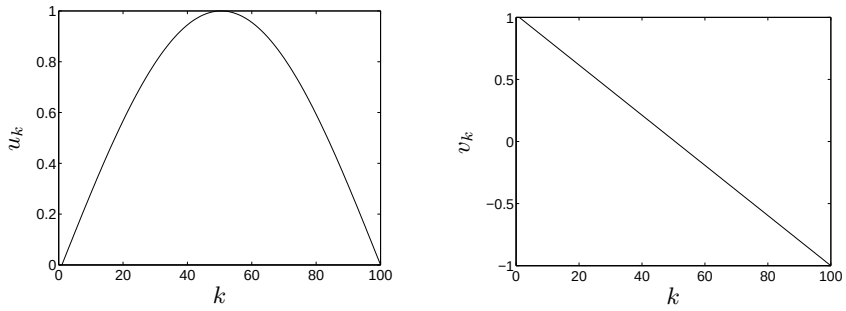
(b) Calculate the angles of v with each of the four signals $x^{(k)}$. This can be done in MATLAB using the command

```
acos((v'*x)/(norm(x)*norm(v)))
```

(which returns the angle between x and v in radians). Which signal $x^{(k)}$ makes the smallest angle with v? Does this confirm your conclusion in part (a)?

1.21 *Multiaccess communication.* A communication channel is shared by several users (transmitters), who use it to send binary sequences (sequences with values $+1$ and -1) to a receiver. The following technique allows the receiver to separate the sequences transmitted by each transmitter. We explain the idea for the case with two transmitters.

We assign to each transmitter a different signal or *code*. The codes are represented as n-vectors u and v: u is the code for user 1, v is the code for user 2. The codes are chosen to be orthogonal ($u^T v = 0$). The figure shows a simple example of two orthogonal codes of length $n = 100$.



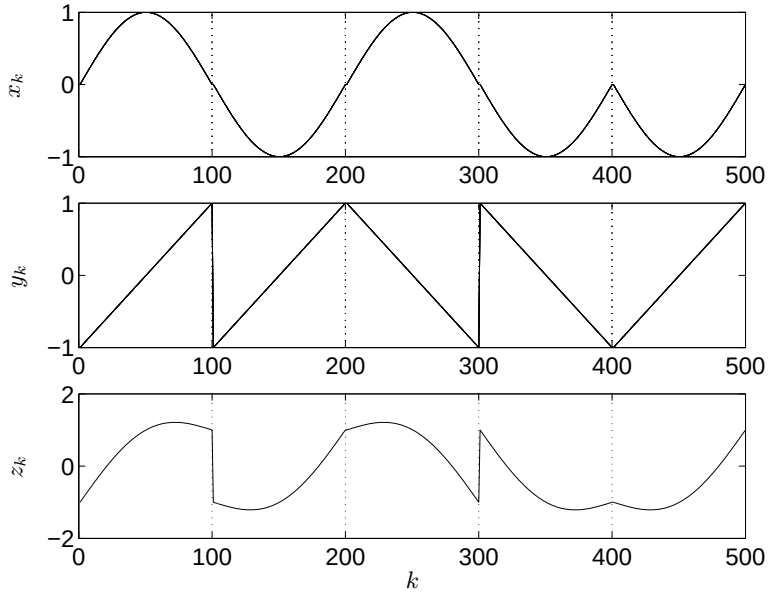
Suppose user 1 wants to transmit a binary sequence $b_1, b_2, \ldots, b_m$ (with values $b_i = 1$ or $b_i = -1$), and user 2 wants to transmit a sequence $c_1, c_2, \ldots, c_m$ (with values $c_i = 1$ or $c_i = -1$). From these sequences and the user codes, we construct two signals x and y, both of length mn, as follows:

$$x = \begin{bmatrix} b_1 u \\ b_2 u \\ \vdots \\ b_m u \end{bmatrix}, \qquad y = \begin{bmatrix} c_1 v \\ c_2 v \\ \vdots \\ c_m v \end{bmatrix}.$$

(Note that here we use block vector notation: x and y consist of m blocks, each of size n. The first block of x is $b_1 u$, the code vector u multiplied with the scalar b_1, etc.) User 1 sends the signal x over the channel, and user 2 sends the signal y. The receiver receives the sum of the two signals. We write the received signal as z:

$$z = x + y = \begin{bmatrix} b_1 u \\ b_2 u \\ \vdots \\ b_m u \end{bmatrix} + \begin{bmatrix} c_1 v \\ c_2 v \\ \vdots \\ c_m v \end{bmatrix}.$$

The figure shows an example where we use the two code vectors u and v shown before. In this example $m = 5$, and the two transmitted sequences are $b = (1, -1, 1, -1, -1)$ and $c = (-1, -1, 1, 1, -1)$.



How can the receiver recover the two sequences b_k and c_k from the received signal z? Let us denote the first block (consisting of the first n values) of the received signal z as $z^{(1)}$: $z^{(1)} = b_1 u + c_1 v$. If we make the inner product of $z^{(1)}$ with u, and use the fact that $u^T v = 0$, we get

$$u^T z^{(1)} = u^T (b_1 u + c_1 v) = b_1 u^T u + c_1 u^T v = b_1 \|u\|^2.$$

Similarly, the inner product with v gives

$$v^T z^{(1)} = v^T (b_1 u + c_1 v) = b_1 v^T u + c_1 v^T v = c_1 \|v\|^2.$$

We see that b_1 and c_1 can be computed from the received signal as

$$b_1 = \frac{u^T z^{(1)}}{\|u\|^2}, \qquad c_1 = \frac{v^T z^{(1)}}{\|v\|^2}.$$

Repeating this for the other blocks of z allows us to recover the rest of the sequences b and c:

$$b_k = \frac{u^T z^{(k)}}{\|u\|^2}, \qquad c_k = \frac{v^T z^{(k)}}{\|v\|^2},$$

if $z^{(k)}$ is the kth block of z.

Download the file `ch1ex21.m` from the class webpage, and run it in MATLAB as `[u,v,z] = ch1ex21`. This generates two code vectors u and v of length $n = 100$, and a received signal z of length $mn = 500$. (The code vectors u and v are different from those used in the figures above.) In addition we added a small noise vector to the received signal z, *i.e.*, we have

$$z = \begin{bmatrix} b_1 u \\ b_2 u \\ \vdots \\ b_m u \end{bmatrix} + \begin{bmatrix} c_1 v \\ c_2 v \\ \vdots \\ c_m v \end{bmatrix} + w$$

where w is unknown but small compared to u and v.

(a) Calculate the angle between the code vectors u and v. Verify that they are nearly (but not quite) orthogonal. As a result, and because of the presence of noise, the formulas for b_k and c_k are not correct anymore. How does this affect the decoding scheme? Is it still possible to compute the binary sequences b and c from z?

(b) Compute $(b_1, b_2, b_3, b_4, b_5)$ and $(c_1, c_2, c_3, c_4, c_5)$.

Chapter 2

Matrices

2.1 Definitions and notation

A *matrix* is a rectangular array of numbers written between brackets, as in

$$A = \begin{bmatrix} 0 & 1 & -2.3 & 0.1 \\ 1.3 & 4 & -0.1 & 0 \\ 4.1 & -1 & 0 & 1.7 \end{bmatrix}.$$

An important attribute of a matrix is its *size* or *dimensions*, *i.e.*, the numbers of rows and columns. The matrix A above has 3 rows and 4 columns, so its size is 3×4. A matrix of size $m \times n$ is called an $m \times n$-matrix.

The *elements* (or *entries* or *coefficients*) of a matrix are the values in the array. The i, j element is the value in the ith row and jth column, denoted by double subscripts: the i, j element of a matrix A is denoted A_{ij} or a_{ij}. The positive integers i and j are called the (row and column, respectively) *indices*. If A is an $m \times n$-matrix, then the row index i runs from 1 to m and the column index j runs from 1 to n. For the example above, $a_{13} = -2.3$, $a_{32} = -1$. The row index of the bottom left element (which has value 4.1) is 3; its column index is 1.

An n-vector can be interpreted as an $n \times 1$-matrix; we do not distinguish between vectors and matrices with one column. A matrix with only one row, *i.e.*, with size $1 \times n$, is sometimes called a *row vector*. As an example,

$$w = \begin{bmatrix} -2.1 & -3 & 0 \end{bmatrix}$$

is a row vector (or 1×3-matrix). A 1×1-matrix is considered to be the same as a scalar.

A *square* matrix has an equal number of rows and columns. A square matrix of size $n \times n$ is said to be of *order* n. A *tall* matrix has more rows than columns (size $m \times n$ with $m > n$). A *wide* matrix has more columns than rows (size $m \times n$ with $n > m$).

All matrices in this course are *real*, *i.e.*, have real elements. The set of real $m \times n$-matrices is denoted $\mathbf{R}^{m \times n}$.

Block matrices and submatrices In some applications it is useful to form matrices whose elements are themselves matrices, as in

$$A = \begin{bmatrix} B & C & D \end{bmatrix}, \qquad E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix},$$

where B, C, D, E_{11}, E_{12}, E_{21}, E_{22} are matrices. Such matrices are called *block matrices*; the elements B, C, and D are called *blocks* or *submatrices*. The submatrices are sometimes named by indices, as in the second matrix E. Thus, E_{11} is the 1,1 block of E.

Of course the block matrices must have the right dimensions to be able to fit together. Matrices in the same (block) row must have the same number of rows (*i.e.*, the same 'height'); matrices in the same (block) column must have the same number of columns (*i.e.*, the same 'width'). In the examples above, B, C, and D must have the same number of rows (*e.g.*, they could be 2×3, 2×2, and 2×1). In the second example, E_{11} must have the same number of rows as E_{12}, and E_{21} must have the same number of rows as E_{22}. E_{11} must have the same number of columns as E_{21}, and E_{12} must have the same number of columns as E_{22}.

As an example, if

$$C = \begin{bmatrix} 0 & 2 & 3 \\ 5 & 4 & 7 \end{bmatrix}, \qquad D = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix},$$

then

$$\begin{bmatrix} C & D \end{bmatrix} = \begin{bmatrix} 0 & 2 & 3 & 2 & 2 \\ 5 & 4 & 7 & 1 & 3 \end{bmatrix}.$$

Using block matrix notation we can write an $m \times n$-matrix A as

$$A = \begin{bmatrix} B_1 & B_2 & \cdots & B_n \end{bmatrix},$$

if we define B_k to be the $m \times 1$-matrix (or m-vector)

$$B_k = \begin{bmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{mk} \end{bmatrix}.$$

Thus, an $m \times n$-matrix can be viewed as an ordered set of n vectors of size m. Similarly, A can be written as a block matrix with one block column and m rows:

$$A = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_m \end{bmatrix},$$

if we define C_k to be the $1 \times n$-matrix

$$C_k = \begin{bmatrix} a_{k1} & a_{k2} & \cdots & a_{kn} \end{bmatrix}.$$

In this notation, the matrix A is interpreted as an ordered collection of m row vectors of size n.

2.2 Zero and identity matrices

A zero matrix is a matrix with all elements equal to zero. The zero matrix of size $m \times n$ is sometimes written as $0_{m\times n}$, but usually a zero matrix is denoted just 0, the same symbol used to denote the number 0 or zero vectors. In this case the size of the zero matrix must be determined from the context.

An identity matrix is another common matrix. It is always square. Its *diagonal* elements, *i.e.*, those with equal row and column index, are all equal to one, and its off-diagonal elements (those with unequal row and column indices) are zero. Identity matrices are denoted by the letter I. Formally, the identity matrix of size n is defined by

$$I_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

Perhaps more illuminating are the examples

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which are the 2×2 and 4×4 identity matrices.

The column vectors of the $n \times n$ identity matrix are the unit vectors of size n. Using block matrix notation, we can write

$$I = \begin{bmatrix} e_1 & e_2 & \cdots & e_n \end{bmatrix}$$

where e_k is the kth unit vector of size n (see section 1.2).

Sometimes a subscript denotes the size of a identity matrix, as in I_4 or $I_{2\times 2}$. But more often the size is omitted and follows from the context. For example, if

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix},$$

then

$$\begin{bmatrix} I & A \\ 0 & I \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 2 & 3 \\ 0 & 1 & 4 & 5 & 6 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The dimensions of the two identity matrices follow from the size of A. The identity matrix in the 1,1 position must be 2×2, and the identity matrix in the 2,2 position must be 3×3. This also determines the size of the zero matrix in the 2,1 position.

The importance of the identity matrix will become clear in section 2.6.

2.3 Matrix transpose

If A is an $m \times n$-matrix, its *transpose*, denoted A^T (or sometimes A'), is the $n \times m$-matrix given by $(A^T)_{ij} = a_{ji}$. In words, the rows and columns of A are transposed

in A^T. For example,

$$\begin{bmatrix} 0 & 4 \\ 7 & 0 \\ 3 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 7 & 3 \\ 4 & 0 & 1 \end{bmatrix}.$$

If we transpose a matrix twice, we get back the original matrix: $(A^T)^T = A$. Note that transposition converts row vectors into column vectors and vice versa.

A square matrix A is *symmetric* if $A = A^T$, *i.e.*, $a_{ij} = a_{ji}$ for all i, j.

2.4 Matrix addition

Two matrices of the same size can be added together. The result is another matrix of the same size, obtained by adding the corresponding elements of the two matrices. For example,

$$\begin{bmatrix} 0 & 4 \\ 7 & 0 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 6 \\ 9 & 3 \\ 3 & 5 \end{bmatrix}.$$

Matrix subtraction is similar. As an example,

$$\begin{bmatrix} 1 & 6 \\ 9 & 3 \end{bmatrix} - I = \begin{bmatrix} 0 & 6 \\ 9 & 2 \end{bmatrix}.$$

(This gives another example where we have to figure out the size of the identity matrix. Since you can only add or subtract matrices of the same size, I refers to a 2×2 identity matrix.)

The following important properties of matrix addition can be verified directly from the definition.

- Matrix addition is commutative: if A and B are matrices of the same size, then $A + B = B + A$.
- Matrix addition is associative: $(A + B) + C = A + (B + C)$. We therefore write both as $A + B + C$.
- $A + 0 = 0 + A = A$. Adding the zero matrix to a matrix has no effect.
- $(A + B)^T = A^T + B^T$. The transpose of a sum of two matrices is the sum of their transposes.

2.5 Scalar-matrix multiplication

Scalar multiplication of matrices is defined as for vectors, and is done by multiplying every element of the matrix by the scalar. For example

$$(-2)\begin{bmatrix} 1 & 6 \\ 9 & 3 \\ 6 & 0 \end{bmatrix} = \begin{bmatrix} -2 & -12 \\ -18 & -6 \\ -12 & 0 \end{bmatrix}.$$

Note that $0\,A = 0$ (where the left-hand zero is the scalar zero).

Several useful properties of scalar multiplication follow directly from the definition. For example, $(\beta A)^T = \beta(A^T)$ for a scalar β and a matrix A. If A is a matrix and β, γ are scalars, then

$$(\beta + \gamma)A = \beta A + \gamma A, \qquad (\beta\gamma)A = \beta(\gamma A).$$

2.6 Matrix-matrix multiplication

It is possible to multiply two matrices using *matrix multiplication*. You can multiply two matrices A and B provided their dimensions are *compatible*, which means the number of columns of A equals the number of rows of B. Suppose A and B are compatible, *e.g.*, A has size $m \times p$ and B has size $p \times n$. Then the product matrix $C = AB$ is the $m \times n$-matrix with elements

$$c_{ij} = \sum_{k=1}^{p} a_{ik}b_{kj} = a_{i1}b_{1j} + \cdots + a_{ip}b_{pj}, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n.$$

There are several ways to remember this rule. To find the i, j element of the product $C = AB$, you need to know the ith row of A and the jth column of B. The summation above can be interpreted as 'moving left to right along the ith row of A' while moving 'top to bottom' down the jth column of B. As you go, you keep a running sum of the product of elements: one from A and one from B.

Matrix-vector multiplication An important special case is the multiplication of a matrix with a vector.

If A is an $m \times n$ matrix and x is an n-vector, then the *matrix-vector product* $y = Ax$ is defined as the matrix-matrix product, with x interpreted as an $n \times 1$-matrix. The result is an m-vector y with elements

$$y_i = \sum_{k=1}^{n} a_{ik}x_k = a_{i1}x_1 + \cdots + a_{in}x_n, \quad i = 1, \ldots, m. \tag{2.1}$$

We can express the result in a number of different ways. If a_k is the kth column of A, then $y = Ax$ can be written

$$y = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = x_1a_1 + x_2a_2 + \cdots + x_na_n.$$

This shows that $y = Ax$ is a linear combination of the columns of A; the coefficients in the linear combination are the elements of x.

The matrix-vector product can also be interpreted in terms of the rows of A. From (2.1) we see that y_i is the inner product of x with the ith row of A:

$$y_i = b_i^T x, \quad i = 1, \ldots, m,$$

if b_i^T is row i of A.

Vector inner product Another important special case is the multiplication of a row vector with a column vector. If a and b are n-vectors, then the inner product

$$a_1b_1 + a_2b_2 + \cdots + a_nb_n$$

can be interpreted as the matrix-matrix product of the $1 \times n$-matrix a^T and the $n \times 1$ matrix b. (This explains the notation a^Tb for the inner product of vectors a and b, defined in section 1.5.)

It is important to keep in mind that $a^Tb = b^Ta$, and that this is very different from ab^T or ba^T: $a^Tb = b^Ta$ is a scalar, while ab^T and ba^T are the $n \times n$-matrices

$$ab^T = \begin{bmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & & \vdots \\ a_nb_1 & a_nb_2 & \cdots & a_nb_n \end{bmatrix}, \qquad ba^T = \begin{bmatrix} b_1a_1 & b_1a_2 & \cdots & b_1a_n \\ b_2a_1 & b_2a_2 & \cdots & b_2a_n \\ \vdots & \vdots & & \vdots \\ b_na_1 & b_na_2 & \cdots & b_na_n \end{bmatrix}.$$

The product ab^T is sometimes called the *outer product* of a with b.

As an exercise on matrix-vector products and inner products, one can verify that if A is $m \times n$, x is an n-vector, and y is an m-vector, then

$$y^T(Ax) = (y^TA)x = (A^Ty)^Tx,$$

i.e., the inner product of y and Ax is equal to the inner product of x and A^Ty.

Properties of matrix multiplication We can now explain the term *identity matrix*. If A is any $m \times n$-matrix, then $AI = A$ and $IA = A$, *i.e.*, when you multiply a matrix by an identity matrix, it has no effect. (Note the different sizes of the identity matrices in the formulas $AI = A$ and $IA = A$.)

Matrix multiplication is (in general) *not commutative:* we *do not* (in general) have $AB = BA$. In fact, BA may not even make sense, or, if it makes sense, may be a different size than AB. For example, if A is 2×3 and B is 3×4, then AB makes sense (the dimensions are compatible) but BA does not even make sense (much less equals AB). Even when AB and BA both make sense and are the same size, *i.e.*, when A and B are square, we do not (in general) have $AB = BA$. As a simple example, take the matrices

$$A = \begin{bmatrix} 1 & 6 \\ 9 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & -1 \\ -1 & 2 \end{bmatrix}.$$

We have

$$AB = \begin{bmatrix} -6 & 11 \\ -3 & -3 \end{bmatrix}, \qquad BA = \begin{bmatrix} -9 & -3 \\ 17 & 0 \end{bmatrix}.$$

Two matrices A and B that satisfy $AB = BA$ are said to *commute*.

The following properties do hold, and are easy to verify from the definition of matrix multiplication.

- Matrix multiplication is associative: $(AB)C = A(BC)$. Therefore we write the product simply as ABC.

- Matrix multiplication is associative with scalar multiplication: $\gamma(AB) = (\gamma A)B$, where γ is a scalar and A and B are matrices (that can be multiplied). This is also equal to $A(\gamma B)$. (Note that the products γA and γB are defined as scalar-matrix products, but in general, unless A and B have one row, not as matrix-matrix products.)
- Matrix multiplication distributes across matrix addition: $A(B+C) = AB + AC$ and $(A+B)C = AC + BC$.
- The transpose of product is the product of the transposes, but in the *opposite* order: $(AB)^T = B^T A^T$.

Row and column interpretation of matrix-matrix product We can derive some additional insight in matrix multiplication by interpreting the operation in terms of the rows and columns of the two matrices.

Consider the matrix product of an $m \times n$-matrix A and an $n \times p$-matrix B, and denote the columns of B by b_k, and the rows of A by a_k^T. Using block-matrix notation, we can write the product AB as

$$AB = A \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix} = \begin{bmatrix} Ab_1 & Ab_2 & \cdots & Ab_p \end{bmatrix}.$$

Thus, the columns of AB are the matrix-vector products of A and the columns of B. The product AB can be interpreted as the matrix obtained by 'applying' A to each of the columns of B.

We can give an analogous row interpretation of the product AB, by partitioning A and AB as block matrices with row vector blocks:

$$AB = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix} B = \begin{bmatrix} a_1^T B \\ a_2^T B \\ \vdots \\ a_m^T B \end{bmatrix} = \begin{bmatrix} (B^T a_1)^T \\ (B^T a_2)^T \\ \vdots \\ (B^T a_m)^T \end{bmatrix}.$$

This shows that the rows of AB are obtained by applying B^T to the transposed row vectors a_k of A.

From the definition of the i, j element of AB in (2.1), we also see that the elements of AB are the inner products of the rows of A with the columns of B:

$$AB = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_p \\ \vdots & \vdots & & \vdots \\ a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_p \end{bmatrix}.$$

Thus we can interpret the matrix-matrix product as the mp inner products $a_i^T b_j$ arranged in an $m \times p$-matrix. When $A = B^T$ this gives the symmetric matrix

$$B^T B = \begin{bmatrix} b_1^T b_1 & b_1^T b_2 & \cdots & b_1^T b_p \\ b_2^T b_1 & b_2^T b_2 & \cdots & b_2^T b_p \\ \vdots & \vdots & \ddots & \vdots \\ b_p^T b_1 & b_p^T b_2 & \cdots & b_p^T b_p \end{bmatrix}.$$

Matrices that can be written as products B^TB for some B are often called *Gram matrices* and arise in many applications. If the set of column vectors $\{b_1, b_2, \ldots, b_p\}$ is *orthogonal* (*i.e.*, $b_i^T b_j = 0$ for $i \neq j$), then the Gram matrix B^TB is diagonal,

$$B^TB = \begin{bmatrix} \|b_1\|_2^2 & 0 & \cdots & 0 \\ 0 & \|b_2\|_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|b_p\|_2^2 \end{bmatrix}.$$

If in addition the vectors are normalized, *i.e.*, $\{b_1, b_2, \ldots, b_p\}$ is an *orthonormal* set of vectors, then $B^TB = I$. The matrix B is then said to be *orthogonal*. (The term *orthonormal* matrix would be more accurate but is less widely used.) In other words, a matrix is orthogonal if its columns form an orthonormal set of vectors.

Matrix powers It makes sense to multiply a square matrix A by itself to form AA. We refer to this matrix as A^2. Similarly, if k is a positive integer, then k copies of A multiplied together is denoted A^k. If k and l are positive integers, and A is square, then $A^kA^l = A^{k+l}$ and $(A^k)^l = A^{kl}$.

Matrix powers A^k with k a negative integer are discussed in the next chapter. Non-integer powers, such as $A^{1/2}$ (the matrix squareroot), are pretty tricky — they might not make sense, or be ambiguous, unless certain conditions on A hold. This is an advanced topic in linear algebra.

2.7 Linear functions

The notation $f : \mathbf{R}^n \to \mathbf{R}^m$ means that f is a function that maps real n-vectors to real m-vectors. Hence, the value of the function f, evaluated at an n-vector x, is an m-vector $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))$. Each of the components f_k of f is itself a scalar valued function of x.

As for scalar valued functions, we sometimes write $f(x) = f(x_1, x_2, \ldots, x_n)$ to emphasize that f is a function of n scalar arguments.

2.7.1 Matrix-vector products and linear functions

Suppose A is an $m \times n$-matrix. We can define a function $f : \mathbf{R}^n \to \mathbf{R}^m$ by $f(x) = Ax$. The linear functions from $\mathbf{R}^n$ to $\mathbf{R}$ discussed in section 1.6 are a special case with $m = 1$.

Superposition and linearity The function $f(x) = Ax$ is *linear*, *i.e.*, it satisfies the superposition property

$$\begin{aligned} f(\alpha x + \beta y) &= A(\alpha x + \beta y) \\ &= A(\alpha x) + A(\beta y) \end{aligned}$$

$$\begin{aligned} &= \alpha(Ax) + \beta(Ay) \\ &= \alpha f(x) + \beta f(y) \end{aligned}$$

for all n-vectors x and y and all scalars α and β. Thus we can associate with every matrix A a linear function $f(x) = Ax$.

Matrix-vector representation of linear functions The converse is also true: if f is a linear function that maps n-vectors to m-vectors, then there exists an $m \times n$-matrix A such that $f(x) = Ax$ for all x. This can be shown in the same way as for scalar valued functions in section 1.6, by showing that if f is linear, then

$$f(x) = x_1 f(e_1) + x_2 f(e_2) + \cdots + x_n f(e_n), \tag{2.2}$$

where e_k is the kth unit vector of size n. The right-hand side can also be written as a matrix-vector product Ax, with

$$A = \begin{bmatrix} f(e_1) & f(e_2) & \cdots & f(e_n) \end{bmatrix}.$$

The expression (2.2) is the same as (1.4), but here $f(x)$ and $f(e_k)$ are vectors. The implications are exactly the same: a linear vector valued function f is completely characterized by evaluating f at the n unit vectors $e_1, \ldots, e_n$.

As in section 1.6 it is easily shown that the matrix-vector representation of a linear function is unique. If $f : \mathbf{R}^{m\times n}$ is a linear function, then there exists exactly one matrix A such that $f(x) = Ax$ for all x.

Examples Below we define five functions f that map n-vectors x to n-vectors $f(x)$. Each function is described in words, in terms of its effect on an arbitrary x.

- f changes the sign of x: $f(x) = -x$.
- f replaces each element of x with its absolute value: $f(x) = (|x_1|, |x_2|, \ldots, |x_n|)$.
- f reverses the order of the elements of x: $f(x) = (x_n, x_{n-1}, \ldots, x_1)$.
- f sorts the elements of x in decreasing order.
- f makes the running sum of the elements in x:

  $$f(x) = (x_1,\ x_1 + x_2,\ x_1 + x_2 + x_3,\ \ldots,\ x_1 + x_2 + \cdots + x_n),$$

  i.e., if $y = f(x)$, then $y_k = \sum_{i=1}^{k} x_i$.

The first function is linear, because it can be expressed as $f(x) = Ax$ with $A = -I$. The second function is not linear. For example, if $n = 2$, $x = (1, 0)$, $y = (0, 0)$, $\alpha = -1$, $\beta = 0$, then

$$f(\alpha x + \beta y) = (1, 0) \neq \alpha f(x) + \beta f(y) = (-1, 0),$$

so superposition does not hold. The third function is linear. To see this it is sufficient to note that $f(x) = Ax$ with

$$A = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix}.$$

(This is the $n \times n$ identity matrix with the order of its columns reversed.) The fourth function is not linear. For example, if $n = 2$, $x = (1, 0)$, $y = (0, 1)$, $\alpha = \beta = 1$, then

$$f(\alpha x + \beta y) = (1, 1) \neq \alpha f(x) + \beta f(y) = (2, 0).$$

The fifth function is linear. It can be expressed as $f(x) = Ax$ with

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix},$$

i.e., $A_{ij} = 1$ if $i \geq j$ and $A_{ij} = 0$ otherwise.

Linear functions and orthogonal matrices Recall that a matrix A is orthogonal if $A^T A = I$. Linear functions $f(x) = Ax$ with A orthogonal possess several important properties.

- f preserves norms:

$$\|f(x)\| = \|Ax\| = ((Ax)^T Ax)^{1/2} = (x^T A^T Ax)^{1/2} = (x^T x)^{1/2} = \|x\|.$$

- f preserves inner products:

$$f(u)^T f(x) = (Au)^T (Ax) = u^T A^T Ax = u^T x.$$

- Combining these two properties, we see that f preserves correlation coefficients: if u and x are nonzero vectors, then

$$\frac{f(u)^T f(x)}{\|f(u)\| \, \|f(x)\|} = \frac{u^T x}{\|u\| \, \|x\|}.$$

  It therefore also preserves angles between vectors.

As an example, consider the function $f(x) = Ax$ where A is the orthogonal matrix

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

Multiplication with A *rotates* a 2-vector x over an angle θ (counter-clockwise if

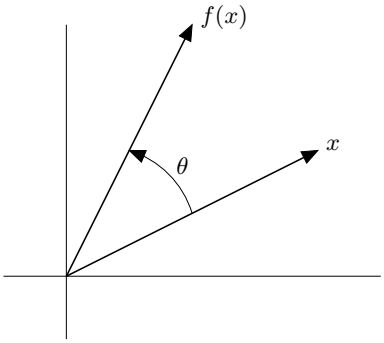


Figure 2.1 Counter-clockwise rotation over an angle θ in $\mathbf{R}^2$.

$\theta \geq 0$, and clockwise if $\theta \leq 0$ (see figure 2.1). This is easily seen by expressing x in polar coordinates, as $x_1 = r\cos\gamma$, $x_2 = r\sin\gamma$. Then

$$\begin{aligned} f(x) &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} r\cos\gamma \\ r\sin\gamma \end{bmatrix} \\ &= \begin{bmatrix} r(\cos\theta\cos\gamma - \sin\theta\sin\gamma) \\ r(\sin\theta\cos\gamma + \cos\theta\sin\gamma) \end{bmatrix} \\ &= \begin{bmatrix} r\cos(\gamma+\theta) \\ r\sin(\gamma+\theta) \end{bmatrix}. \end{aligned}$$

Given this interpretation, it is clear that this function f preserves norms and angles of vectors.

Matrix-matrix products and composition Suppose A is an $m \times n$-matrix and B is $n \times p$. We can associate with these matrices two linear functions $f : \mathbf{R}^n \to \mathbf{R}^m$ and $g : \mathbf{R}^p \times \mathbf{R}^n$, defined as $f(x) = Ax$ and $g(x) = Bx$. The composition of the two functions is the function $h : \mathbf{R}^p \times \mathbf{R}^n$ with

$$h(x) = f(g(x)) = ABx.$$

This is a linear function, that can be written as $h(x) = Cx$ with $C = AB$.

Using this interpretation it is easy to justify why in general $AB \neq BA$, even when the dimensions are compatible. Evaluating the function $h(x) = ABx$ means we first evaluate $y = Bx$, and then $z = Ay$. Evaluating the function BAx means we first evaluate $y = Ax$, and then $z = By$. In general, the order matters.

As an example, take the 2×2-matrices

$$A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

for which

$$AB = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \qquad BA = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

The mapping $f(x) = Ax = (-x_1, x_2)$ changes the sign of the first element of the vector x. The mapping $g(x) = Bx = (x_2, x_1)$ reverses the order of two elements of x. If we evaluate $ABx = (-x_2, x_1)$, we first reverse the order, and then change the sign of the first element. This result is obviously different from $BAx = (x_2, -x_1)$, obtained by changing the sign of the first element, and then reversing the order of the elements.

Affine functions A vector valued function $f : \mathbf{R}^n \to \mathbf{R}^m$ is called affine if it can be expressed as $f(x) = Ax + b$, with A an $m \times n$-matrix and b and m-vector. It can be shown that a function $f : \mathbf{R}^n \to \mathbf{R}^m$ is affine if and only if

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

for all n-vectors x, y, and all scalars α, β that satisfy $\alpha + \beta = 1$.

The matrix A and the vector b in the representation of an affine function as $f(x) = Ax + b$ are unique. These parameters can be obtained by evaluating f at the vectors 0, e_1, $\ldots$, e_n, where e_k is the kth unit vector in $\mathbf{R}^n$. We have

$$A = \begin{bmatrix} f(e_1) - f(0) & f(e_2) - f(0) & \cdots & f(e_n) - f(0) \end{bmatrix}, \qquad b = f(0).$$

2.7.2 Linearization of a vector valued function

Suppose $f : \mathbf{R}^n \to \mathbf{R}^m$ is continuously differentiable, *i.e.*, the mn partial derivatives

$$\frac{\partial f_i(\hat{x})}{\partial x_j} = \lim_{t \to 0} \frac{f_i(\hat{x} + te_j) - f_i(\hat{x})}{t}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n,$$

exist at every $\hat{x}$, and they are continuous functions of $\hat{x}$. We can collect the partial derivatives in an $m \times n$-matrix

$$Df(\hat{x}) = \begin{bmatrix} \partial f_1(\hat{x})/\partial x_1 & \partial f_1(\hat{x})/\partial x_2 & \cdots & \partial f_1(\hat{x})/\partial x_n \\ \partial f_2(\hat{x})/\partial x_1 & \partial f_2(\hat{x})/\partial x_2 & \cdots & \partial f_2(\hat{x})/\partial x_n \\ \vdots & \vdots & & \vdots \\ \partial f_m(\hat{x})/\partial x_1 & \partial f_m(\hat{x})/\partial x_2 & \cdots & \partial f_m(\hat{x})/\partial x_n \end{bmatrix}.$$

This is called the *Jacobian* or *derivative matrix* of f at $\hat{x}$.

The *first-order Taylor approximation* or *linearization* of f around $\hat{x}$ is defined as a function $f_{\text{aff}} : \mathbf{R}^n \to \mathbf{R}^m$, with

$$f_{\text{aff}}(x) = f(\hat{x}) + Df(\hat{x})(x - \hat{x}). \tag{2.3}$$

This is an affine function of x, since it can be expressed as $f_{\text{aff}}(x) = Ax + b$ with $A = Df(\hat{x})$ and $b = f(\hat{x}) - Df(\hat{x})\hat{x}$. The linearization f_{aff} gives a good approximation of $f(x)$ for x close to $\hat{x}$, *i.e.*, $f_{\text{aff}}(x) \approx f(x)$ for $x \approx \hat{x}$. This property is often expressed in terms of the x and y deviations $\delta x = x - \hat{x}$ and $\delta y = f(x) - f(\hat{x})$. These deviations satisfy

$$\delta y \approx Df(\hat{x})\delta x$$

for small δx.

In section 1.6.2 we defined the gradient of a scalar valued function $\mathbf{R}^n \to \mathbf{R}$ as the n-vector of its first partial derivatives. The derivative matrix of a vector function $f : \mathbf{R}^n \to \mathbf{R}^m$ can be expressed in terms of the gradients of the components f_k of f as

$$Df(\hat{x}) = \begin{bmatrix} \nabla f_1(\hat{x})^T \\ \nabla f_2(\hat{x})^T \\ \vdots \\ \nabla f_m(\hat{x})^T \end{bmatrix}. \tag{2.4}$$

The linearization of the vector function f in (2.3) gives in matrix notation the same result as

$$f_{\mathrm{aff},i}(\hat{x}) = f_i(\hat{x}) + \nabla f_i(\hat{x})^T(x - \hat{x}), \quad i = 1, \ldots, m,$$

the first-order approximations of the m components of f (see (1.5)).

2.8 Matrix norm

The norm of a matrix serves the same purpose as the norm of a vector. It is a measure of the size or magnitude of the matrix. As for vectors, many possible definitions exist. For example, in analogy with the Euclidean norm of a vector x,

$$\|x\| = (x_1^2 + x_2^2 + \cdots + x_n^2)^{1/2},$$

we might define the norm of an $m \times n$-matrix A as

$$(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2)^{1/2},$$

i.e., the square root of the sum of the squares of the elements of A. This is called the *Frobenius norm* of A.

In this course, we will use a different definition of matrix norm (known as spectral norm or 2-norm). The norm of an $m \times n$-matrix A, denoted $\|A\|$, is defined as

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}. \tag{2.5}$$

Note that we use the same notation for the matrix norm as for the vector norm. For example, in (2.5), x and Ax are vectors, so $\|x\|$ and $\|Ax\|$ refer to (Euclidean) vector norms; A is a matrix, so $\|A\|$ refers to the matrix norm. We will see later that if A is a matrix with one column (so it can be interpreted as a vector or a matrix), then the two norms are equal.

To better understand the definition of $\|A\|$, it is useful to recall the 'operator' or 'function' interpretation of a matrix A: we can associate with an $m \times n$-matrix A a linear function

$$f(x) = Ax$$

that maps n-vectors x to m-vectors $y = Ax$. For each nonzero x, we can calculate the norm of x and Ax, and refer to the ratio $\|Ax\|/\|x\|$ as the *gain* or *amplification factor* of the operator f in the direction of x:

$$\mathbf{gain}(x) = \frac{\|Ax\|}{\|x\|}.$$

Of course, the gain generally depends on x, and might be large for some vectors x and small (or zero) for others. The matrix norm, as defined in (2.5), is the maximum achievable gain, over all possible choices of x:

$$\|A\| = \max_{x \neq 0} \mathbf{gain}(x) = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

Although it is not yet clear how we can actually compute $\|A\|$ using this definition, it certainly makes sense as a measure for the magnitude of A. If $\|A\|$ is small, say, $\|A\| \ll 1$, then $\|Ax\| \ll \|x\|$ for all $x \neq 0$, which means that the function f strongly attenuates the input vectors x. If $\|A\|$ is large, then there exist input vectors x for which $\mathbf{gain}(x)$ is large.

Simple examples The norm of simple matrices can be calculated directly by applying the definition. For example, if $A = 0$, then $Ax = 0$ for all x, so $\|Ax\|/\|x\| = 0$ for all x, and hence $\|A\| = 0$. If $A = I$, we have $Ax = x$, and hence

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{x \neq 0} \frac{\|x\|}{\|x\|} = 1.$$

As another example, suppose

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}.$$

We have $Ax = (x_2, -x_3, x_1)$, hence

$$\|Ax\| = \sqrt{x_2^2 + x_3^2 + x_1^2} = \|x\|,$$

so this matrix also has norm one:

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{x \neq 0} \frac{\|x\|}{\|x\|} = 1.$$

Next, assume that A is an $m \times 1$-matrix,

$$A = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}.$$

In this case x is a scalar, so $\|x\| = |x|$ and

$$Ax = \begin{bmatrix} a_1 x \\ a_2 x \\ \vdots \\ a_m x \end{bmatrix}, \qquad \|Ax\| = |x|\sqrt{a_1^2 + a_2^2 + \cdots + a_m^2}.$$

Therefore $\|Ax\|/\|x\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_m^2}$ for all nonzero x, and

$$\|A\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_m^2}.$$

The matrix norm of a matrix with one column is equal to the Euclidean norm of the column vector.

In these four examples, the ratio $\|Ax\|/\|x\|$ is the same for all nonzero x, so maximizing over x is trivial. As an example where the gain varies with x, we consider a diagonal matrix

$$A = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_n \end{bmatrix}.$$

Here the gain is not independent of x. For example, for $x = (1, 0, \ldots, 0)$, we have $Ax = (\alpha_1, 0, \ldots, 0)$, so

$$\mathbf{gain}(x) = \frac{\|Ax\|}{\|x\|} = \frac{|\alpha_1|}{1} = |\alpha_1|.$$

If $x = (0, 1, 0, \ldots, 0)$, the gain is $|\alpha_2|$, etc. To find the matrix norm, we have to find the value of x that maximizes the gain. For general x, we have $Ax = (\alpha_1 x_1, \alpha_2 x_2, \ldots, \alpha_n x_n)$, and therefore

$$\|A\| = \max_{x \neq 0} \frac{\sqrt{\alpha_1^2 x_1^2 + \alpha_2^2 x_2^2 + \cdots + \alpha_n^2 x_n^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}}.$$

We will show that this is equal to the maximum of the absolute values of the diagonal elements α_i:

$$\|A\| = \max\{|\alpha_1|, |\alpha_2|, \ldots, |\alpha_n|\}.$$

Suppose for simplicity that

$$\alpha_1^2 \geq \alpha_2^2 \geq \cdots \geq \alpha_n^2,$$

so $|\alpha_1| = \max_i |\alpha_i|$. Then

$$\alpha_1^2 x_1^2 + \alpha_2^2 x_2^2 + \cdots + \alpha_n^2 x_n^2 \leq \alpha_1^2 (x_1^2 + x_2^2 + \cdots + x_n^2)$$

for all $x \neq 0$, and therefore

$$\frac{\sqrt{\alpha_1^2 x_1^2 + \alpha_2^2 x_2^2 + \cdots + \alpha_n^2 x_n^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} \leq |\alpha_1|.$$

Moreover for $x = (1, 0, \ldots, 0)$, we have

$$\frac{\sqrt{\alpha_1^2 x_1^2 + \alpha_2^2 x_2^2 + \cdots + \alpha_n^2 x_n^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} = |\alpha_1|$$

so it follows that

$$\max_{x \neq 0} \frac{\sqrt{\alpha_1^2 x_1^2 + \alpha_2^2 x_2^2 + \cdots + \alpha_n^2 x_n^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} = |\alpha_1| = \max_{i=1,\ldots,n} |\alpha_i|.$$

Properties of the matrix norm The following useful properties of the matrix norm follow from the definition. We leave the proofs as an exercise.

- (Homogeneity) $\|\beta A\| = |\beta| \, \|A\|$.
- (Triangle inequality) $\|A + B\| \leq \|A\| + \|B\|$.
- (Definiteness) $\|A\| \geq 0$ for all A and $\|A\| = 0$ if and only if $A = 0$.
- $\|A\| = \max_{\|x\|=1} \|Ax\|$.
- $\|Ax\| \leq \|A\| \, \|x\|$ for all vectors x (that can be multiplied with A).
- $\|AB\| \leq \|A\| \, \|B\|$ for all matrices B (that can be multiplied with A).
- $\|A\| = \|A^T\|$ (see exercise 2.15).

Computing the norm of a matrix The simple examples given above are meant to illustrate the definition of matrix norm, and not to suggest a practical method for calculating the matrix norm. In fact, except for simple matrices, it is very difficult to see which vector x maximizes $\|Ax\|/\|x\|$ and it is usually impossible to find the norm 'by inspection' or a simple calculation.

In practice, however, there exist efficient and reliable *numerical* methods for calculating the norm of a matrix. In MATLAB the command is `norm(A)`. Algorithms for computing the matrix norm are based on techniques that are not covered in this course. For our purposes it is sufficient to know that the norm of a matrix is readily computed.

Exercises

Block matrix notation

2.1 *Block matrix notation.* Consider the block matrix

$$A = \begin{bmatrix} I & B & 0 \\ B^T & 0 & 0 \\ 0 & 0 & BB^T \end{bmatrix}$$

where B is 10×5. What are the dimensions of the zero matrices and the identity matrix in the definition of A, and of A itself?

2.2 Block matrices can be transposed, added, and multiplied as if the elements were numbers, provided the corresponding elements have the right sizes (*i.e.*, 'conform') and you are careful about the order of multiplication. As an example, take a block matrix

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

where B is $m \times p$, C is $m \times q$, D is $n \times p$ and E is $n \times q$.

(a) Show that

$$A^T = \begin{bmatrix} B^T & D^T \\ C^T & E^T \end{bmatrix}.$$

(b) If γ is a scalar, show that

$$\gamma A = \begin{bmatrix} \gamma B & \gamma C \\ \gamma D & \gamma E \end{bmatrix}.$$

(c) Suppose

$$Z = \begin{bmatrix} V & W \\ X & Y \end{bmatrix}$$

where V $m \times p$, W is $m \times q$, X is $n \times p$ and Y is $n \times q$. Show that

$$A + Z = \begin{bmatrix} B + V & C + W \\ D + X & E + Y \end{bmatrix}.$$

(d) Suppose

$$Z = \begin{bmatrix} X \\ Y \end{bmatrix}$$

where X is $p \times r$ and Y is $q \times r$. Show that

$$AZ = \begin{bmatrix} BX + CY \\ DX + EY \end{bmatrix}.$$

Matrix-vector product and linear functions

2.3 *Shift matrices.*

(a) Give a simple description in words of the function $f(x) = Ax$, where A is the 5×5-matrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

(b) Same question for $f(x) = A^k x$, for $k = 2, 3, 4, 5$. (A^k is the kth power of A: $A^2 = AA$, $A^3 = A^2A = AAA$, etc.)

(c) Same question for $f(x) = A^T x$.

2.4 *Permutation matrices.* A square matrix A is called a *permutation matrix* if it satisfies the following three properties:

- all elements of A are either zero or one
- each column of A contains exactly one element equal to one
- each row of A contains exactly one element equal to one.

The matrices

$$A_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

are examples of permutation matrices. A less formal definition is the following: a permutation matrix is the identity matrix with its rows reordered.

(a) Let A be an $n \times n$ permutation matrix. Give a simple description in words of the relation between a n-vector x and $f(x) = Ax$.

(b) We can also define a second linear function $g(x) = A^T x$, in terms of the same permutation matrix. What is the relation between g and f?
Hint. Consider the composition $g(f(x))$. First try a few simple examples with $n = 2$ and $n = 3$, and then generalize.

2.5 The *cross-product* $a \times x$ of two 3-vectors $a = (a_1, a_2, a_3)$ and $x = (x_1, x_2, x_3)$ is defined as the vector

$$a \times x = \begin{bmatrix} a_2x_3 - a_3x_2 \\ a_3x_1 - a_1x_3 \\ a_1x_2 - a_2x_1 \end{bmatrix}.$$

(a) Assume a is fixed and nonzero. Show that the function $f(x) = a \times x$ is a linear function of x, by giving a matrix A that satisfies $f(x) = Ax$ for all x.

(b) Verify that $A^TA = (a^Ta)I - aa^T$.

(c) Show that for nonzero x,

$$\|a \times x\| = \|a\| \, \|x\| \, |\sin\theta|$$

where θ is the angle between a and x.

2.6 *Projection of a vector x on a given vector y.* Let y be a given n-vector, and consider the function $f : \mathbf{R}^n \to \mathbf{R}^n$, defined as

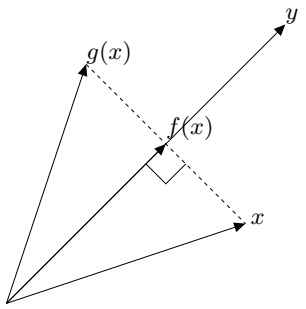
$$f(x) = \frac{x^Ty}{\|y\|^2} y.$$

From exercise 1.10 we know that $f(x)$ is the projection of x on the (fixed, given) vector y. Is f a linear function of x? If your answer is yes, give an $n \times n$-matrix A such that $f(x) = Ax$ for all x. If your answer is no, show with an example that f does not satisfy the definition of linearity $f(\alpha u + \beta v) = \alpha f(u) + \beta f(v)$.

2.7 *Reflection of a vector about a given vector y.* As in problem 2.6, let $f(x)$ be the projection of x on a given n-vector y. Define $g : \mathbf{R}^n \to \mathbf{R}^n$ as the reflection of x with respect to the line defined by y. $g(x)$ can be expressed as

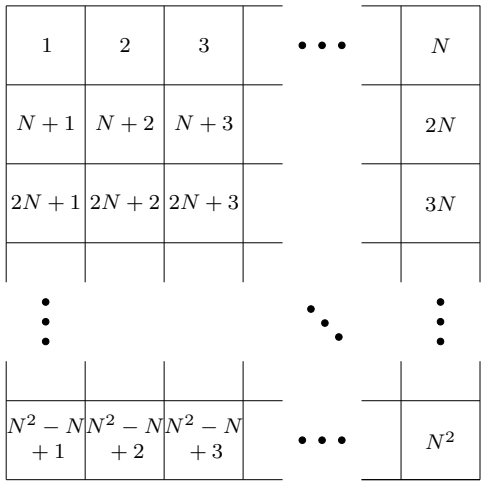
$$g(x) = x + 2(f(x) - x)$$

The definition is illustrated in the figure below.



Show that g is linear, and that it can be expressed as $g(x) = Ax$ with A an orthogonal matrix.

2.8 *Linear functions of grayscale images.* The figure shows an image divided into $N \times N$ pixels. We represent the image as a vector x of size N^2, where x_i denotes the grayscale level of pixel i (a number between 0 and 1, where 0 represents black, and 1 represents white).



Each of the following operations defines a function $y = f(x)$, where the N^2-vector x represents the original image, and the N^2-vector y represents the resulting image. For each of these operations, show that f is linear, and give an $N^2 \times N^2$-matrix A such that

$$y = Ax$$

for all x. You can assume that $N = 3$ for simplicity.

(a) Turn the image represented by x upside-down.

(b) Rotate the image clockwise over 90°.

(c) Translate the image up by 1 pixel, and to the right by 1 pixel. In the translated image, assign a value $y_i = 0$ to the pixels in the first column and the last row.

(d) Replace each pixel grayscale level by the average of the grayscale levels in a 3×3-neighborhood.

2.9 Represent each of the following three functions $f : \mathbf{R}^2 \to \mathbf{R}^2$ as a matrix-vector product $f(x) = Ax$.

(a) $f(x)$ is obtained by reflecting x about the x_1 axis.

(b) $f(x)$ is x reflected about the x_1 axis, followed by a counterclockwise rotation of 30 degrees.

(c) $f(x)$ is x rotated counterclockwise over 30 degrees, followed by a reflection about the x_1 axis.

2.10 We consider n-vectors x that represent signals, with x_k the value of the signal at time k for $k = 1, \ldots, n$. Below we describe two linear functions of x that produce new signals $f(x)$. For each function, give a matrix A such that $f(x) = Ax$ for all x.

(a) $2\times$ *downsampling.* We assume n is even, and define $f(x)$ as the $n/2$-vector y with elements $y_k = x_{2k}$. To simplify your notation you can assume that $n = 8$, *i.e.*,

$$f(x) = (x_2,\, x_4,\, x_6,\, x_8).$$

(b) $2\times$ *up-conversion with linear interpolation.* We define $f(x)$ as the $(2n-1)$-vector y with elements $y_k = x_{(k+1)/2}$ if k is odd and $y_k = (x_{k/2} + x_{k/2+1})/2$ if k is even. To simplify your notation you can assume that $n = 5$, *i.e.*,

$$f(x) = (x_1,\, \frac{x_1+x_2}{2},\, x_2,\, \frac{x_2+x_3}{2},\, x_3,\, \frac{x_3+x_4}{2},\, x_4,\, \frac{x_4+x_5}{2},\, x_5).$$

2.11 *Linear dynamical system.* A linear dynamical system is described by the recurrence

$$x(t+1) = Ax(t) + Bu(t), \quad t = 0, 1, 2, \ldots$$

The vector $x(t) \in \mathbf{R}^n$ is the *state* at time t, and $u(t) \in \mathbf{R}^m$ is the *input* at time t. The parameters in the model are the $n \times n$-matrix A and the $n \times m$-matrix B.

Find a formula for $x(N)$ (for a given $N \geq 0$) in terms of $x(0)$, $u(0)$, $\ldots$, $u(N-1)$. Is $x(t)$ a linear function of $x(0)$, $u(0), \ldots, u(N-1)$? If not, give a specific counterexample. If it is linear, find a specific matrix G of size n by $n + mN$ such that

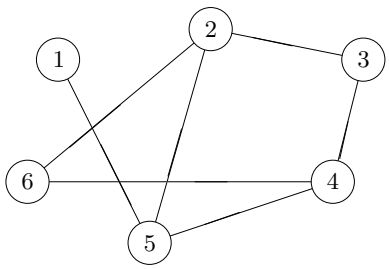
$$x(N) = G \begin{bmatrix} x(0) \\ u(0) \\ \vdots \\ u(N-1) \end{bmatrix}$$

for all $x(0)$, $u(0)$, $\ldots$, $u(N-1)$.

Matrix-matrix product

2.12 Consider an undirected graph with n nodes, and no self loops (*i.e.*, all edges connect two different nodes). Define an $n \times n$-matrix A with elements

$$a_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are linked by an edge} \\ 0 & \text{otherwise.} \end{cases}$$



The matrix A for the graph in the figure is

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Note that $A = A^T$, and $a_{ii} = 0$ since there are no self loops.

Let $B = A^k$ denote the kth power of the matrix A, where k is an integer greater than or equal to one. Give a simple interpretation of the elements B_{ij} of B in terms of the original graph.

Hint. The answer is related to the concept of a *path* of length m between nodes i and j. This is defined as a sequence of nodes $r_0, r_1, \ldots, r_m$, such that $r_0 = i$, $r_m = j$, and for each $k = 0, \ldots, m-1$, there is an edge between r_k and r_{k+1}. For example, you can think of the nodes in the graph as airports, and of the edges as direct flights connecting two airports. A path of length m between nodes i and j is a route between airports i and j that requires exactly m flights.

Matrix norms

2.13 As mentioned on page 40, there is no simple formula that expresses the matrix norm

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|,$$

as an explicit function of the elements of A. In special cases however, such as the examples listed in section 2.8, we can easily determine a vector x that maximizes the gain $\|Ax\|/\|x\|$, and hence calculate the matrix norm.

What are the norms of the following matrices A?

(a) A matrix with one row:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \end{bmatrix}.$$

(b) A matrix of the form $A = uu^T$ where u is a given n-vector.

(c) A matrix of the form $A = uv^T$ where u and v are given n-vectors.

2.14 Compute the matrix norm of each of the following matrices, without using MATLAB.

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & -3/2 \end{bmatrix}, \quad \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & -3/2 \end{bmatrix}.$$

2.15 In this exercise we show that $\|A\| = \|A^T\|$.

(a) Let u be a vector. Show that

$$\|u\| = \max_{v \neq 0} \frac{u^T v}{\|v\|}.$$

(b) Let A be a matrix. Show that

$$\|A\| = \max_{y \neq 0, x \neq 0} \frac{y^T A x}{\|x\|\|y\|}.$$

(c) Use the result of part (b) to show that $\|A\| = \|A^T\|$.

Chapter 3

Linear equations

3.1 Introduction

This chapter covers much of the matrix theory needed for the algorithms in chapters 5 through 10.

Linear equations The chapter centers around sets of linear equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m. \end{aligned}$$

This is the general form of m linear equations in n variables or unknowns $x_1, \ldots, x_n$. The mn coefficients a_{ij} and the m right-hand sides b_i are given parameters. Note that we allow $m \neq n$.

The equations can be written succinctly in matrix notation as

$$Ax = b \tag{3.1}$$

where the coefficient matrix A is $m \times n$ with elements a_{ij}, the variable x is an n-vector, and the right-hand side b is an m-vector.

The equations are *overdetermined* if $m > n$, *underdetermined* if $m < n$, and *square* if $m = n$. A set of equations with zero right-hand side, $Ax = 0$, is called a *homogeneous* set of equations.

Examples If $m = n = 1$, A is a scalar and there are three cases to distinguish. If $A \neq 0$, the solution is unique and equal to $x = b/A$. If $A = 0$ and $b \neq 0$, the equation does not have a solution. If $A = 0$ and $b = 0$, every real number x is a solution, *i.e.*, the solution set is the real line $\mathbf{R}$.

We encounter the same three cases in higher dimensions. A set of linear equations can have a unique solution, as in the example

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Here $x = (1, 1, -1)$ is the only solution. The equations can be *unsolvable*, as in the example

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

There can be infinitely many solutions, as for example

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}.$$

Here $x = (1, 1, \alpha)$ is a solution for all α.

Matrix inverses A key role will be played by *matrix inversion*, used to generalize the operation $x = b/A$ to higher dimensions.

The division $\frac{B}{A}$ of two matrices is not defined (except when they are 1×1) and it is easy to understand why. For scalar A and B, the division can be interpreted in two equivalent ways, as $A^{-1}B$ or BA^{-1}. To make sense of the division in the matrix case, one first has to define A^{-1} (what are the dimensions, when does it exist, is it unique, et cetera). Even when the inverse A^{-1} is well defined and unique, matrix division remains ambiguous, because it can mean multiplying with the inverse on the left or on the right. As we saw in chapter 2 the distinction between left and right multiplication matters because the matrix product is not commutative ($XY \neq YX$ in general).

We therefore need to distinguish between two types of inverses. A matrix X that satisfies

$$AX = I$$

is called a *right inverse* of A. The matrix A is said to be *right-invertible* if a right inverse exists. A matrix X that satisfies

$$XA = I$$

is called a *left inverse* of A. The matrix A is *left-invertible* if a left inverse exists.

In general, a right-invertible matrix can have more than one right inverse and a left-invertible matrix can have more than one left inverse. However we can immediately note one important property. If a matrix is left- *and* right-invertible, then the left and right inverses are unique and equal: if $AX = I$ and $YA = I$, then

$$X = (YA)X = Y(AX) = Y.$$

In this case we call the matrix $X = Y$ simply *the inverse* of A, denoted A^{-1}, and say that A is *invertible*.

Inverse and linear equations Several simple but important facts about left and right inverses and the solution of $Ax = b$ follow directly from the definitions.

- If A has a right inverse C, then $x = Cb$ is a solution of $Ax = b$, since

  $$Ax = A(Cb) = (AC)b = b.$$

  Therefore, if A is right-invertible, then the linear equation $Ax = b$ has *at least one* solution for every value of b.

- If A has a left inverse D and x is a solution of $Ax = b$, then multiplying the two sides of $Ax = b$ on the left with D gives

  $$x = DAx = Db.$$

  This means that x is the only solution, and equal to Db. Therefore, if A is left-invertible, then the linear equation $Ax = b$ has *at most one* solution for every value of b (*i.e.*, if a solution exists, it is unique).

- By combining these two observations, we conclude that if A is invertible, then $Ax = b$ has a *unique* solution for every value of b. The solution can be written as $x = A^{-1}b$.

3.2 Range and nullspace

Conditions for existence and uniquess of solutions of $Ax = b$ can be stated abstractly in terms of two fundamental subspaces associated with A, its range and its nullspace.

3.2.1 Range

The *range* of an $m \times n$-matrix A, denoted $\mathbf{range}(A)$, is the set of all m-vectors that can be written as Ax:

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

The range of a matrix is a *subspace* of $\mathbf{R}^m$, *i.e.*, it has the property that all linear combinations of vectors in $\mathbf{range}(A)$ also belong to $\mathbf{range}(A)$. In other words, if u, v in $\mathbf{range}(A)$, then

$$\alpha u + \beta v \in \mathbf{range}(A)$$

for all scalars α, β. This is straightforward to verify. If $u, v \in \mathbf{range}(A)$ then there exist vectors x, y with $u = Ax$, $v = Ay$. A linear combination $w = \alpha u + \beta v$ can be expressed as

$$w = \alpha u + \beta v = \alpha(Ax) + \beta(Ay) = A(\alpha x + \beta y).$$

Therefore w is in $\mathbf{range}(A)$.

The range of a matrix always contains the zero vector (take $x = 0$ in the definition). When $\mathbf{range}(A) = \mathbf{R}^m$ we say that A has a *full range*.

Example The range of the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 2 \\ 0 & -1 & -1 \end{bmatrix} \tag{3.2}$$

contains the vectors $(x_1 + x_3, x_1 + x_2 + 2x_3, -x_2 - x_3)$ for all possible values of x_1, x_2, x_3. The same set can be expressed more simply as

$$\mathbf{range}(A) = \{(u, u + v, -v) \mid u, v \in \mathbf{R}\}.$$

This matrix A does not have a full range. One can check that the vector $y = (1, 0, 0)$, for example, does not belong to the range.

Existence of solutions By definition of range, the linear equation $Ax = b$ is solvable if and only if $b \in \mathbf{range}(A)$. If A has a full range, then $Ax = b$ is solvable for all values of b. If A does not have a full range, then there exist values of b for which $Ax = b$ is not solvable.

3.2.2 Nullspace

The *nullspace* of an $m \times n$-matrix A, denoted $\mathbf{nullspace}(A)$, is the set of n-vectors for which the matrix-vector product with A is zero:

$$\mathbf{nullspace}(A) = \{x \in \mathbf{R}^n \mid Ax = 0\}.$$

The nullspace is a subspace of $\mathbf{R}^n$: if x, y in $\mathbf{nullspace}(A)$ then

$$\alpha x + \beta y \in \mathbf{nullspace}(A)$$

for all scalars α, β, because $A(\alpha x + \beta y) = \alpha Ax + \beta Ay = 0$ if $Ax = Ay = 0$.

Note that the nullspace always contains the zero vector. When $\mathbf{nullspace}(A) = \{0\}$, we say that A has a *zero nullspace*.

Examples A vector $x = (x_1, x_2, x_3)$ is in the nullspace of the matrix A in (3.2) if

$$x_1 + x_3 = 0, \qquad x_1 + x_2 + 2x_3 = 0, \qquad -x_1 - x_3 = 0.$$

This means that $x_1 = x_2 = -x_3$. Therefore the nullspace is

$$\mathbf{nullspace}(A) = \{(u, u, -u) \mid u \in \mathbf{R}\}.$$

As another example, consider the matrix

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^{n-1} \end{bmatrix}.$$

This is called a Vandermonde matrix, and arises in polynomial interpolation (see exercise 3.1). We claim that if the numbers t_i are distinct ($t_i \neq t_j$ for $i \neq j$), then A has a zero nullspace. To show this, we work out the product Ax:

$$\begin{aligned} Ax &= \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \begin{bmatrix} x_1 + x_2 t_1 + x_3 t_1^2 + \cdots + x_n t_1^{n-1} \\ x_1 + x_2 t_2 + x_3 t_2^2 + \cdots + x_n t_2^{n-1} \\ \vdots \\ x_1 + x_2 t_n + x_3 t_n^2 + \cdots + x_n t_n^{n-1} \end{bmatrix}. \end{aligned}$$

Therefore $Ax = 0$ means that

$$x_1 + x_2 t_i + x_2 t_i^2 + \cdots + x_n t_i^{n-1} = 0$$

for $i = 1, 2, \ldots, n$. In other words the polynomial

$$p(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}$$

has n distinct real roots t_1, t_2, …, t_n. This is impossible unless $x_1 = x_2 = \cdots = x_n = 0$, because a polynomial of degree $n-1$ cannot have more than $n-1$ distinct real roots. Therefore, for this matrix, $Ax = 0$ implies $x = 0$.

Uniqueness of solution If A has a zero nullspace, then the equation $Ax = b$ has at most one solution (in other words, if a solution exists, it is unique). Suppose x and y are two solutions ($Ax = b$ and $Ay = b$). Then

$$A(x - y) = Ax - Ay = 0,$$

so $x - y$ is in the nullspace of A. If the nullspace is $\{0\}$, we must have $x = y$.

If the nullspace of A contains nonzero vectors and $Ax = b$ is solvable, then it has infinitely many solutions. If x is a solution and v is a nonzero vector in the nullspace, then

$$A(x + \alpha v) = Ax + \alpha Av = Ax = b$$

for all scalars α. Therefore $x + \alpha v$ is a solution for any value of α.

3.3 Nonsingular matrices

In this section we derive a fundamental result in linear algebra: a *square* matrix has a full range if and only if it has a zero nullspace. In other words, for a square matrix A, the full range and zero nullspace and properties are equivalent:

$$A \text{ has a full range} \quad \Longleftrightarrow \quad A \text{ has a zero nullspace.} \tag{3.3}$$

Later we will see that the equivalence only holds for square matrices; a rectangular matrix can have a zero nullspace or a full range, but not both.

A square matrix with a zero nullspace (equivalently, a full range) is called *nonsingular* or *regular*. The opposite is a *singular* matrix. If A is $n \times n$ and singular, then $\mathbf{range}(A) \neq \mathbf{R}^n$ and $\mathbf{nullspace}(A) \neq \{0\}$.

3.3.1 Schur complement

We will need the following definition. Let A be an $n \times n$-matrix, partitioned as

$$A = \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{3.4}$$

with a_{11} a scalar, A_{21} of size $(n-1) \times 1$, A_{12} of size $1 \times (n-1)$, and A_{22} square of order $n-1$. If $a_{11} \neq 0$ then the matrix

$$S = A_{22} - \frac{1}{a_{11}} A_{21} A_{12} \tag{3.5}$$

is called the *Schur complement* of a_{11} in A.

Variable elimination Schur complements arise when eliminating a variable in a set of linear equations. Suppose the coefficient matrix in $Ax = b$ is partitioned as in (3.4) and x and b are partitioned accordingly:

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ B_2 \end{bmatrix}. \tag{3.6}$$

Here x_1 and b_1 are scalars, and X_2 and B_2 are $(n-1)$-vectors. If $a_{11} \neq 0$, we can solve for x_1 from the first equation $a_{11}x_1 + A_{12}X_2 = b_1$. The answer is

$$x_1 = \frac{1}{a_{11}}(b_1 - A_{12}X_2). \tag{3.7}$$

Substituting x_1 in the left-hand side of the second equation gives

$$\begin{aligned} A_{21}x_1 + A_{22}X_2 &= \frac{1}{a_{11}} A_{21}(b_1 - A_{12}X_2) + A_{22}X_2 \\ &= (A_{22} - \frac{1}{a_{11}} A_{21}A_{12})X_2 + \frac{b_1}{a_{11}} A_{21} \\ &= SX_2 + \frac{b_1}{a_{11}} A_{21}. \end{aligned}$$

After the substitution the second equation therefore reduces to the following equation in X_2:

$$SX_2 = B_2 - \frac{b_1}{a_{11}} A_{21}. \tag{3.8}$$

We conclude that if $a_{11} \neq 0$, the set of n linear equations (3.6) is equivalent to the set of $n-1$ equations (3.8), combined with the assignment (3.7) for x_1.

Full range Schur complement The preceding observations on variable elimination lead to the following result. If $a_{11} \neq 0$, then

$$A \text{ has a full range} \iff S \text{ has a full range.} \tag{3.9}$$

This can be verified as follows. If S has a full range, then (3.8) is solvable for all B_2 and b_1. The solution X_2, combined with x_1 in (3.7), satisfies (3.6), which therefore has a solution for all b_1, B_2. Hence A has a full range.

Conversely, if A has a full range, then (3.6) has a solution for all b_1, B_2. In particular, if we take $b_1 = 0$, $B_2 = d$, then (3.8) reduces to $SX_2 = d$. Therefore this equation is solvable for all d, *i.e.*, S has a full range.

Zero nullspace Schur complement A similar result holds for the nullspace. If $a_{11} \neq 0$ then

$$A \text{ has a zero nullspace} \iff S \text{ has a zero nullspace.} \tag{3.10}$$

To see this, consider the equation (3.6) with zero right-hand side:

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{3.11}$$

By variable elimination, this is equivalent to (3.7) and (3.8) with $b_1 = 0$, $B_2 = 0$:

$$SX_2 = 0, \qquad x_1 = -\frac{1}{a_{11}} A_{12} X_2. \tag{3.12}$$

Therefore if X_2 is a nonzero element of the nullspace of S, then (x_1, X_2) with $x_1 = -(1/a_{11})A_{12}X_2$ is a nonzero element of the nullspace of A. Conversely, if (x_1, X_2) is a nonzero element in the nullspace of A, then from (3.12) we see that $X_2 \neq 0$ and X_2 is in the nullspace of S. In other words, the nullspace of A contains nonzero elements if and only the nullspace of S contains nonzero elements.

3.3.2 Equivalence of zero nullspace and full range property

We are now ready to show the main result (3.3). The proof is by induction on the order of the matrix, which means the following. We first check that the result holds for $n = 1$. Then we show that if it holds for $n = k$ it also holds for $n = k+1$. From this we conclude that it holds for all n.

The base part is trivial. If $n = 1$, 'full range' and 'zero nullspace' both simply mean $A \neq 0$.

For the induction part, we assume that (3.3) holds for all square matrices of order k, and show that it holds for a matrix A of order $n = k + 1$. We distinguish three cases.

Nonzero leading element If $a_{11} \neq 0$, we partition A as in (3.6) with A_{22} square of order k. From the properties of the Schur complement in the previous section:

$$\begin{aligned} A \text{ has a full range} &\iff S \text{ has a full range} \\ &\iff S \text{ has a zero nullspace} \\ &\iff A \text{ has a zero nullspace.} \end{aligned}$$

The first line is (3.9). The second line follows from the induction assumption, since S is a square matrix of order k. The third line is (3.10).

Zero leading element, nonzero first column or row If $a_{11} = 0$ but the first column and the first row of A are not entirely zero, we can argue as follows.

If there is a nonzero element in position $i > 1$ of the first column of A, we swap rows 1 and i of A and call the new matrix $\tilde{A}$. For this matrix $\tilde{a}_{11} \neq 0$. The reordering of the rows of a matrix does not change the nullspace ($Ax = 0$ if and only if $\tilde{A}x = 0$). It also does not change the dimension of the range, *i.e.*, if $Ax = b$ is solvable for all b, then $\tilde{A}x = \tilde{b}$ is solvable for all $\tilde{b}$.

If the entire first column of A is zero, but there is a nonzero element in position $j > 1$ of the first row we swap columns 1 and j of A and call the resulting matrix $\tilde{A}$. Again, $\tilde{a}_{11} \neq 0$. Reordering the columns does not change the range of a matrix or the dimension of its nullspace.

Thus we can write

$$\begin{aligned} A \text{ has a full range} &\iff \tilde{A} \text{ has a full range} \\ &\iff \tilde{A} \text{ has a zero nullspace} \\ &\iff A \text{ has a zero nullspace.} \end{aligned}$$

The second step follows because $\tilde{A}$ has a nonzero leading element $\tilde{a}_{11}$, so this is the case proved in the previous paragraph.

Zero first row and column Finally, suppose the entire first row and first column of A are zero. Then A has a nonzero nullspace, because $Ax = 0$ for $x = e_1 = (1, 0, \ldots, 0)$. The matrix A also does not have a full range, since all vectors $y = Ax$ must have a zero first element, so $y = e_1$ is not in the range. We conclude that in this case (3.3) holds because the assertions on the two sides of the implication are false.

3.3.3 Inverse

In this section we show that the square invertible matrices are exactly the nonsingular matrices.

Nonsingular matrices are invertible Recall from the beginning of section 3.3 that a nonsingular matrix is defined as a square matrix with a full range and a zero nullspace. Equivalently, $Ax = b$ has a unique solution x for every right-hand side b.

We first show that if A is nonsingular, then it has a right inverse, *i.e.*, there exists a matrix X that satisfies

$$AX = I.$$

The equation $AX = I$ is equivalent to

$$A \begin{bmatrix} X_1 & X_2 & \cdots & X_n \end{bmatrix} = \begin{bmatrix} e_1 & e_2 & \cdots & e_n \end{bmatrix},$$

where X_i denotes the ith column of X and e_i is the ith unit vector. The matrix equation $AX = I$ is therefore equivalent to n sets of linear equations

$$AX_1 = e_1, \qquad AX_2 = e_2, \qquad \ldots, \qquad AX_n = e_n.$$

Since A is nonsingular, each of these n equations has exactly one solution X_i. Therefore there is exactly one matrix X that satisfies $AX = I$.

The next argument shows that $XA = I$, *i.e.*, X is also a left inverse. Since $AX = I$ we have $AXA = A$ and therefore $A(XA - I) = 0$. This implies

$$XA = I$$

because A has a zero nullspace.

We conclude that for a nonsingular matrix, left and right inverses exist, and that they are identical and unique. We refer to X simply as *the inverse* of A, denoted A^{-1}. Nonsingular matrices are also called *invertible*.

Singular matrices are not invertible For completeness, we briefly discuss singular matrices and explain why a singular matrix is not right-invertible or left-invertible.

As noted at the end of the introduction (section 3.1), if A is right-invertible, with a right inverse C, then every vector y can be expressed as $y = Ax$ by choosing $x = Cy$. A right-invertible matrix therefore has a full range. Since a singular matrix does not have a full range, it cannot be right-invertible.

If A is left-invertible, with a left inverse D, then $Ax = 0$ implies $x = DAx = 0$. A left-invertible matrix therefore has a zero nullspace. Since a singular matrix does not have a zero nullspace, it cannot be left-invertible.

Inverse of transpose If A is nonsingular with inverse $X = A^{-1}$, then

$$X^T A^T = (AX)^T = I, \qquad A^T X^T = (XA)^T = I.$$

Therefore A^T is nonsingular with inverse

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

Since the order of the transpose and inverse operations does not matter, we write this matrix simply as A^{-T}.

Inverse of matrix product If A and B are nonsingular and of the same dimension, then their product AB is nonsingular with inverse $B^{-1}A^{-1}$. This follows from

$$(B^{-1}A^{-1})(AB) = B^{-1}(A^{-1}A)B = B^{-1}B = I,$$

and

$$(AB)(B^{-1}A^{-1}) = A(BB^{-1})A^{-1} = AA^{-1} = I.$$

By repeatedly applying this property we can give a meaning to negative integer powers of a nonsingular matrix A. We have

$$(A^k)^{-1} = (A^{-1})^k$$

and denote this matrix as A^{-k}. For example, $A^{-2} = A^{-1}A^{-1} = (AA)^{-2}$.

If we define A^0 as $A^0 = I$, then $A^{k+l} = A^k A^l$ for all integers k and l.

3.3.4 Square sets of linear equations

To conclude the discussion of nonsingular matrices, we summarize the implications of the results for the solution of square sets of linear equations $Ax = b$.

If A is nonsingular, then the equation $Ax = b$ has a unique solution for every value of b. At least one solution exists because A has a full range. At most one solution exists because A has a zero nullspace. By multiplying the two sides of $Ax = b$ on the left with A^{-1} we can express the solution as

$$x = A^{-1}b.$$

If A is singular then the equation never has a unique solution. Either the solution set is empty (if b is not in the range of A) or there are infinitely many solutions (if b is in the range of A).

3.4 Positive definite matrices

Positive definite matrices will be important in our discussion of left and right inverses in section 3.5. They also arise frequently in applications.

3.4.1 Definition

A square matrix A is *positive definite* if it is symmetric ($A = A^T$) and

$$x^T Ax > 0 \text{ for all nonzero } x.$$

In other words, $x^T Ax \geq 0$ for all x, and $x^T Ax = 0$ only if $x = 0$. A slightly larger class are the *positive semidefinite* matrices: a matrix A is positive semidefinite if it is symmetric and

$$x^T Ax \geq 0 \text{ for all } x.$$

All positive definite matrices are positive semidefinite but not vice-versa, because the definition of positive definite matrix includes the extra requirement that $x = 0$ is the only vector for which $x^T Ax = 0$.

Examples Practical methods for checking positive definiteness of a matrix will be discussed in chapter 6. For small matrices, one can directly apply the definition and examine the sign of the product $x^T Ax$. It is useful to note that for a general square matrix A, the product $x^T Ax$ can be expanded as

$$x^T Ax = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j,$$

i.e., $x^T Ax$ is a sum of all cross-products $x_i x_j$ multiplied with the i, j element of A. Each product $x_i x_j$ for $i \neq j$ appears twice in the sum, and if A is symmetric

the coefficients of the two terms are equal ($a_{ij} = a_{ji}$). Therefore, for a symmetric matrix,

$$x^T Ax = \sum_{i=1}^{n} a_{ii} x_i^2 + 2 \sum_{i=1}^{n} \sum_{j=1}^{i-1} a_{ij} x_i x_j.$$

The first sum is over the diagonal elements of A; the second sum is over the strictly lower-triangular elements.

For example, with

$$A = \begin{bmatrix} 9 & 6 \\ 6 & 5 \end{bmatrix}$$

we get $x^T Ax = 9x_1^2 + 12x_1x_2 + 5x_2^2$. To check the sign of $x^T Ax$, we complete the squares:

$$\begin{aligned} x^T Ax &= 9x_1^2 + 12x_1x_2 + 5x_2^2 \\ &= (3x_1 + 2x_2)^2 - 4x_2^2 + 5x_2^2 \\ &= (3x_1 + 2x_2)^2 + x_2^2. \end{aligned}$$

Clearly $x^T Ax \geq 0$ for all x. Moreover $x^T Ax = 0$ only if $x_1 = x_2 = 0$. Therefore $x^T Ax > 0$ for all nonzero x and we conclude that A is positive definite.

The matrix

$$A = \begin{bmatrix} 9 & 6 \\ 6 & 3 \end{bmatrix}, \tag{3.13}$$

on the other hand, is not positive semidefinite, because

$$\begin{aligned} x^T Ax &= 9x_1^2 + 12x_1x_2 + 3x_2^2 \\ &= (3x_1 + 2x_2)^2 - 4x_2^2 + 3x_2^2 \\ &= (3x_1 + 2x_2)^2 - x_2^2. \end{aligned}$$

From this expression it is easy to find values of x for which $x^T Ax < 0$. For example, for $x = (-2/3, 1)$, we get $x^T Ax = -1$.

The matrix

$$A = \begin{bmatrix} 9 & 6 \\ 6 & 4 \end{bmatrix}$$

is positive semidefinite because

$$\begin{aligned} x^T Ax &= 9x_1^2 + 12x_1x_2 + 4x_2^2 \\ &= (3x_1 + 2x_2)^2 \end{aligned}$$

and this is nonnegative for all x. A is not positive definite because $x^T Ax = 0$ for some nonzero x, *e.g.*, $x = (2, -3)$.

3.4.2 Properties

Diagonal The diagonal elements of a positive definite matrix are all positive. This is easy to show from the definition. If A is positive definite, then $x^T Ax > 0$ for all nonzero x. In particular, $x^T Ax > 0$ for $x = e_i$ (the ith unit vector). Therefore,

$$e_i^T Ae_i = a_{ii} > 0$$

for $i = 1, \ldots, n$. Positivity of the diagonal elements is a *necessary* condition for positive definitess of A, but it is far from sufficient. For example, the matrix defined in (3.13) is not positive definite, although it has positive diagonal elements.

Schur complement The Schur complement of the $1,1$ element of a positive definite matrix is positive definite. To see this, consider a positive definite matrix A of order n. Partition A as

$$A = \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}$$

where a_{11} is a scalar, A_{21} has size $(n-1) \times 1$, and A_{22} is a symmetric matrix of order $n-1$. (Since A is symmetric, we wrote its $1,2$ block as the transpose of the $2,1$ block.) The Schur complement of a_{11} is

$$S = A_{22} - \frac{1}{a_{11}} A_{21} A_{21}^T,$$

and is defined because $a_{11} \neq 0$. To show that S is positive definite, pick any nonzero $(n-1)$-vector v. Then

$$x = \begin{bmatrix} -(1/a_{11}) A_{21}^T v \\ v \end{bmatrix}$$

is a nonzero n-vector and

$$\begin{aligned} x^T A x &= \begin{bmatrix} -(1/a_{11}) v^T A_{21} & v^T \end{bmatrix} \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} -(1/a_{11}) A_{21}^T v \\ v \end{bmatrix} \\ &= \begin{bmatrix} -(1/a_{11}) v^T A_{21} & v^T \end{bmatrix} \begin{bmatrix} 0 \\ Sv \end{bmatrix} \\ &= v^T S v. \end{aligned}$$

Since A is positive definite, we have $x^T A x = v^T S v > 0$ for all nonzero v. Therefore S is positive definite.

Positive definite matrices are nonsingular Positive definite matrices have zero nullspaces: if $Ax = 0$ then $x^T A x = 0$ and if A is positive definite, this implies $x = 0$.

From the theory of section 3.3, we can conclude that a positive definite matrix has a full range, that it is nonsingular, and that it is invertible, since these properties are all equivalent for square matrices.

3.4.3 Gram matrices

In chapter 2 (page 32) we defined a *Gram matrix* as a matrix of the form $B^T B$. All Gram matrices are positive semidefinite since

$$x^T B^T B x = (Bx)^T (Bx) = \|Bx\|^2 \geq 0 \tag{3.14}$$

for all x. This holds regardless of the values and dimensions of B. A Gram matrix is not necessarily positive definite and (3.14) shows what the required properties of B are: the matrix B^TB is positive definite if $\|Bx\| = 0$ implies $x = 0$. In other words, B^TB is positive definite if and only if B has a zero nullspace:

$$B^TB \text{ is positive definite} \iff B \text{ has a zero nullspace.} \tag{3.15}$$

To this we can add another useful characterization, in terms of the range of B^T:

$$B^TB \text{ is positive definite} \iff B^T \text{ has a full range.} \tag{3.16}$$

The proof goes as follows. First assume that B^TB is positive definite (hence, invertible; see section 3.4). Then B^T has a full range because every vector y can be expressed as $y = B^Tx$ by choosing $x = B(B^TB)^{-1}y$.

Conversely, assume B^T has a full range. Let y be a vector for which $y^TB^TBy = \|By\|^2 = 0$. Since B^T has full range, we can express y as $y = B^Tx$ for some x. Taking inner products with y, we get

$$y^Ty = y^TB^Tx = (By)^Tx = 0,$$

because $By = 0$. Thus, $y^TB^TBy = 0$ implies $y = 0$. Therefore B^TB is positive definite.

Finally, note that (3.15) and (3.16) in tandem prove a fundamental result in linear algebra:

$$B \text{ has a zero nullspace} \iff B^T \text{ has a full range.}$$

3.4.4 Singular positive semidefinite matrices

We have seen that positive definite matrices are nonsingular. We now show that if A is positive semidefinite but not positive definite, then it is singular.

Assume that A is positive semidefinite ($x^TAx \geq 0$ for all x) but not positive definite, *i.e.*, there exists a nonzero x with $x^TAx = 0$. We will show that x is in the nullspace of A. Since A has a nonzero nullspace, it is singular.

Take an arbitrary y and define the function

$$f(t) = (x + ty)^TA(x + ty)$$

where t is a scalar variable. We have $f(t) \geq 0$ for all t because A is positive semidefinite. Expanding the products in the definition of f and using $x^TAx = 0$, we obtain

$$\begin{aligned} f(t) &= x^TAx + tx^TAy + ty^TAx + t^2y^TAy \\ &= 2ty^TAx + t^2y^TAy. \end{aligned}$$

The function f is quadratic in t, with $f(0) = 0$. Since $f(t) \geq 0$ for all t the derivative of f at $t = 0$ must be zero, *i.e.*, $f'(0) = 2y^TAx = 0$. Since y is arbitrary, this is only possible if $Ax = 0$.

3.5 Left- and right-invertible matrices

Recall that a matrix A is left-invertible if it has a left inverse (a matrix X that satisfies $XA = I$). A matrix A is right-invertible if it has a right inverse (a matrix X that satisfies $AX = I$).

Left-invertible matrices We establish the following important property:

$$A \text{ is left-invertible} \iff A \text{ has a zero nullspace.} \tag{3.17}$$

We first show the '$\Rightarrow$' part. Assume A is left-invertible and that X is a left inverse ($XA = I$). Then $Ax = 0$ implies $x = XAx = 0$. Therefore A has a zero nullspace.

To show the '$\Leftarrow$' part, we assume A has a zero nullspace. From (3.15) this means that A^TA is positive definite and hence invertible. Then a simple calculation shows that the matrix

$$X = (A^TA)^{-1}A^T \tag{3.18}$$

is a left inverse: $XA = (A^TA)^{-1}(A^TA) = I$. Therefore A is left-invertible.

If A is left-invertible, then it always has the particular left inverse (3.18). This matrix is called the *Moore-Penrose pseudoinverse* of a left-invertible matrix, or simply the *pseudoinverse*. In general, however, other left inverses exist.

If A is square, the expression for the pseudoinverse can be simplified using the rules for the inverses of transposes and products:

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

Note however that this simplification is only correct if A is square. For nonsquare matrices, it is wrong to distribute the inverse in $(A^TA)^{-1}$.

Right-invertible matrices The counterpart of (3.17) for right-invertible matrices is the property

$$A \text{ is right-invertible} \iff A \text{ has a full range.} \tag{3.19}$$

To show the '$\Rightarrow$' part, note that if X is a right inverse of A, then every y can be expressed as $y = Ax$ by choosing $x = Xy$. Hence A has a full range.

The '$\Leftarrow$' part follows from (3.16). If A has a full range, then AA^T is positive definite and invertible, so the matrix

$$X = A^T(AA^T)^{-1} \tag{3.20}$$

exists. Multiplying with A on both sides shows that X is a right inverse: $AX = (AA^T)(AA^T)^{-1} = I$.

The specific right inverse (3.20) is called the (Moore-Penrose) pseudoinverse of a right-invertible matrix.

Dimensions Left- and right-invertibility impose constraints on the dimensions of a matrix. A left-invertible matrix is necessarily square or tall, *i.e.*, if A is $m \times n$ and left-invertible then $m \geq n$. This is easily seen as follows. Assume $m < n$ and A has a left inverse X. By partitioning X and A we can write $I = XA$ as

$$\begin{bmatrix} I_m & 0 \\ 0 & I_{n-m} \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \begin{bmatrix} A_1 & A_2 \end{bmatrix} = \begin{bmatrix} X_1A_1 & X_1A_2 \\ X_2A_1 & X_2A_2 \end{bmatrix},$$

where X_1 and A_1 are $m \times m$, X_2 is $(n-m) \times m$, and A_2 is $m \times (n-m)$. If we examine the four blocks in the equations we quickly reach a contradiction. Since A_1 and X_1 are square, the $1,1$ block $I = X_1A_1$ implies that A_1 is nonsingular with $A_1^{-1} = X_1$. Multiplying the two sides of $X_1A_2 = 0$ on the left with A_1 then gives $A_2 = A_1X_1A_2 = 0$. However this contradicts the $2,2$ block $X_2A_2 = I$.

In a similar way we can show that a right-invertible matrix is necessarily square or wide (*i.e.*, $n \geq m$).

Orthogonal matrices In section 2.6 we have defined an orthogonal matrix as a matrix A that satisfies $A^TA = I$. In the terminology of this chapter, A is orthogonal if its transpose is a left inverse. From the discussion of the dimensions of left-invertible matrices in the previous paragraph, we know that only tall or square matrices can be orthogonal. An example is

$$A = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{6} \\ 1/\sqrt{3} & 1/\sqrt{6} \\ -1/\sqrt{3} & \sqrt{2/3} \end{bmatrix}.$$

If A is orthogonal and square, then $A^TA = I$ means that A^T is the inverse of A and therefore also a right inverse. For square orthogonal matrices we therefore have $A^TA = AA^T = I$. It is important to note that this last property only holds for *square* orthogonal matrices. If A is tall ($m > n$) and orthogonal then $A^TA = I$ but $AA^T \neq I$.

3.6 Summary

We have studied relations between ten matrix properties.

1. A has a zero nullspace.
2. A^T has a full range.
3. A has a left inverse.
4. A^TA is positive definite.
5. $Ax = b$ has at most one solution for every value of b.
6. A has a full range.

7. A^T has a zero nullspace.
8. A has a right inverse.
9. AA^T is positive definite.
10. $Ax = b$ has at least one solution for every value of b.

Square matrices For a square matrix the ten properties are equivalent. If A satisfies any one of these properties, it also satisfies the other nine. Moreover the left and right inverses are unique and equal, and referred to as the inverse of A, written A^{-1}.

Square matrices that satisfy these properties are called nonsingular or invertible. If A is nonsingular, then $Ax = b$ has a unique solution $x = A^{-1}b$ for every value of b.

Tall matrices If A has more rows than columns, then the properties 1–5 are equivalent. Matrices that satisfy these properties are called left-invertible. Properties 6–10 never hold for a tall matrix.

Wide matrices If A has more columns than rows, then the properties 6–10 are equivalent. Matrices that satisfy these properties are called right-invertible. Properties 1–5 never hold for a wide matrix.

Exercises

Formulating linear equations

3.1 *Polynomial interpolation.* In this problem we construct polynomials

$$p(t) = x_1 + x_2 t + \cdots + x_{n-1} t^{n-2} + x_n t^{n-1}$$

of degree 5, 10, and 15 (*i.e.*, for $n = 6$, 11, 16), that interpolate points on the graph of the function $f(t) = 1/(1 + 25t^2)$ in the interval $[-1, 1]$. For each value of n, we compute the interpolating polynomial as follows. We first generate $n + 1$ pairs (t_i, y_i), using the MATLAB commands

```
t = linspace(-1,1,n)';
y = 1./(1+25*t.^2);
```

This produces two vectors: a vector `t` with n elements t_i, equally spaced in the interval $[-1, 1]$, and a vector `y` with elements $y_i = f(t_i)$. (See '`help rdivide`' and '`help power`' for the meaning of the operations `./` and `.^`.) We then solve a set of linear equations

$$\begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-2} & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-2} & t_2^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & t_{n-1} & \cdots & t_{n-1}^{n-2} & t_{n-1}^{n-1} \\ 1 & t_n & \cdots & t_n^{n-2} & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} \tag{3.21}$$

to find the coefficients x_i.

Calculate the three polynomials (for $n = 6$, $n = 11$, $n = 16$). Plot the three polynomials and the function f on the interval $[-1, 1]$, and attach a printout of the plots to your solutions. What do you conclude about the effect of increasing the degree of the interpolating polynomial?

MATLAB hints.

- To construct the coefficient matrix in (3.21), you can write a double for-loop, or use the MATLAB command `vander(t)`, which constructs a matrix of the form

$$\begin{bmatrix} t_1^{n-1} & t_1^{n-2} & \cdots & t_1^2 & t_1 & 1 \\ t_2^{n-1} & t_2^{n-2} & \cdots & t_2^2 & t_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ t_n^{n-1} & t_n^{n-2} & \cdots & t_n^2 & t_n & 1 \end{bmatrix}.$$

  Type '`help vander`' for details. This is almost exactly what we need, but you have to 'flip' this matrix from left to right. This operation is also built in in MATLAB (type `help fliplr`).
- Use `x=A\b` to solve a set of linear equations $Ax = b$.
- We are interested in the behavior of the interpolating polynomials between the points t_i that you used in the construction. Therefore, when you plot the three polynomials, you should use a much denser grid of points (*e.g.*, 200 points equally spaced in interval $[-1, 1]$) than the n points that you used to generate the polynomials.

3.2 Formulate the following problem as a set of linear equations. Compute two cubic polynomials

$$p(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3, \qquad q(t) = d_0 + d_1 t + d_2 t^2 + d_3 t^3$$

that satisfy the following conditions:

(a) $p(t_1) = y_1$, $p(t_2) = y_2$, $p(t_3) = y_3$

(b) $q(t_5) = y_5$, $q(t_6) = y_6$, $q(t_7) = y_7$

(c) $p(t_4) = q(t_4)$, $p'(t_4) = q'(t_4)$.

The variables in the problem are the 8 coefficients c_i, d_i. The numbers t_i, y_i are given, with

$$t_1 < t_2 < t_3 < t_4 < t_5 < t_6 < t_7.$$

Condition (a) specifies the value of $p(t)$ at $t = t_1$, t_2, t_3. Condition (b) specifies the value of $q(t)$ at $t = t_5$, t_6, t_7. Condition (c) specifies that at $t = t_4$ the polynomials p and q should have the same value and the same derivative.

Test the method in MATLAB, on test data created as follows. We take the 7 points t_i equally spaced in the interval $[-0.75, 0.25]$, and choose $y_i = 1/(1 + 25t_i^2)$ for $i = 1, \ldots, 7$:

```
t = linspace(-0.75,0.25,7)';
y = 1./(1+25*t.^2);
```

(We actually do not need y_4, because we do not require the two polynomials to satisfy $p(t_4) = y_4$.) Calculate the two polynomials $p(t)$ and $q(t)$, and plot them on the interval $[-0.75, 0.25]$. Attach a printout of the plots to your solutions. (For more MATLAB hints, see the remarks in problem 3.1.)

3.3 Express the following problem as a set of linear equations. Find a cubic polynomial

$$f(t) = c_1 + c_2(t - t_1) + c_3(t - t_1)^2 + c_4(t - t_1)^2(t - t_2)$$

that satisfies

$$f(t_1) = y_1, \qquad f(t_2) = y_2, \qquad f'(t_1) = s_1, \qquad f'(t_2) = s_2.$$

The numbers t_1, t_2, y_1, y_2, s_1, s_2 are given, with $t_1 \neq t_2$. The unknowns are the coefficients c_1, c_2, c_3, c_4. Write the equations in matrix-vector form $Ax = b$, and solve them.

3.4 Express the following problem as a set of linear equations. Find a rational function

$$f(t) = \frac{c_0 + c_1 t + c_2 t^2}{1 + d_1 t + d_2 t^2}$$

that satisfies the following conditions

$$f(1) = 2.3, \qquad f(2) = 4.8, \qquad f(3) = 8.9, \qquad f(4) = 16.9, \qquad f(5) = 41.0.$$

The variables in the problem are the coefficients c_0, c_1, c_2, d_1 and d_2. Write the equations in matrix-vector form $Ax = b$ and solve the equations with MATLAB.

3.5 Express the following problem as a set of linear equations. Find a quadratic function

$$f(u_1, u_2) = \begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + r$$

that satisfies the following six conditions:

$$f(0, 1) = 6, \qquad f(1, 0) = 6, \qquad f(1, 1) = 3,$$

$$f(-1, -1) = 7, \qquad f(1, 2) = 2, \qquad f(2, 1) = 6.$$

The variables in the problem are the parameters p_{11}, p_{12}, p_{22}, q_1, q_2 and r. Write the equations in matrix-vector form $Ax = b$, and solve the equations with MATLAB.

3.6 Let $a = (a_1, a_2, a_3)$, $b = (b_1, b_2, b_3)$, $c = (c_1, c_2, c_3)$, $d = (d_1, d_2, d_3)$ be four given points in $\mathbf{R}^3$. The points do not lie in one plane. Algebraically, this can be expressed by saying that the vectors $b-a$, $c-a$, $d-a$ are linearly independent, *i.e.*, $y_1(b-a)+y_2(c-a)+y_3(d-a) = 0$ only if $y_1 = y_2 = y_3 = 0$.

Suppose we are given the distances of a point $x = (x_1, x_2, x_3)$ to the four points:

$$\|x - a\| = r_a, \qquad \|x - b\| = r_b, \qquad \|x - c\| = r_c, \qquad \|x - d\| = r_d.$$

Write a set of linear equations $Ax = f$, with A nonsingular, from which the coordinates x_1, x_2, x_3 can be computed. Explain why the matrix A is nonsingular.

3.7 A simple population model for two interacting species is given by the following two equations:

$$\begin{aligned} x_1(t+1) &= ax_1(t) + bx_1(t)x_2(t) \\ x_2(t+1) &= cx_2(t) + dx_1(t)x_2(t), \end{aligned}$$

where $x_1(t)$, $x_2(t)$ are the populations of the two species at the beginning of a certain period (*e.g.*, one year), and $x_1(t+1)$, $x_2(t+1)$ are the populations at the end of the period. For example, in a predator-prey system where species 1 is the prey and species 2 is the predator, we would have $a > 1$ (the prey population grows in the absence of predators), $c < 1$ (the predator population declines in the absence of prey), $b < 0$ and $d > 0$ (if we assume that the frequency of encounters between the two species is proportional to $x_1(t)x_2(t)$).

Suppose you are given observations of $x_1(t)$, $x_2(t)$, over N periods $t = 1, \ldots, t = N$. You are asked to estimate the four coefficients a, b, c, d in the model, based on the observations. Express this problem as a set of linear equations. Write the equations in matrix-vector form $Ax = b$. What are the dimensions of A?

3.8 *Numerical integration.* Numerical integration rules (also called *quadrature rules*) are numerical methods for approximating integrals

$$\int_a^b f(t)\, dt.$$

The general idea is to approximate the integral by a weighted sum of function values $f(t_i)$, at n points t_i in the interval $[a, b]$:

$$\int_a^b f(t)\, dt \approx \sum_{i=1}^n w_i f(t_i),$$

where $a \leq t_1 < t_2 < \cdots < t_n \leq b$. The points t_i are called the *abscissas* or *nodes* of the integration rule, and the coefficients w_i are the *weights*. Many different integration rules exist, which use different choices of n, w_i, and t_i.

For simplicity we will take $a = 0$, $b = 1$.

(a) Suppose you are given n distinct points t_i in the interval $[0, 1]$, and you are asked to determine the weights w_i in such a way that the integration rule is exact for the functions $f(t) = t^k$, $k = 0, 1, \ldots, n-1$. In other words, we require that

$$\int_0^1 f(t)\, dt = \sum_{i=1}^n w_i f(t_i)$$

if f is one of the functions

$$f(t) = 1, \qquad f(t) = t, \qquad f(t) = t^2, \qquad \ldots, \qquad f(t) = t^{n-1}.$$

Formulate this problem as a set of linear equations with variables $w_1, \ldots, w_n$.

(b) Suppose you use the weights computed in (a). Show that the integration rule is exact for all polynomials of degree $n-1$ or less. In other words, show that

$$\int_0^1 f(t)\,dt = \sum_{i=1}^n w_i f(t_i)$$

if $f(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_{n-1}t^{n-1}$.

(c) Apply your method for part (a) to compute the weights in the following integration rules.

(a) $n = 2$, $t_1 = 0$, $t_2 = 1$. This gives the *trapezoid rule*.
(b) $n = 3$, $t_1 = 0$, $t_2 = 0.5$, $t_3 = 1$. This gives *Simpson's rule*.
(c) $n = 4$, $t_1 = 0$, $t_2 = 1/3$, $t_3 = 2/3$, $t_4 = 1$.

Compare the accuracy of these three integration rules when applied to the function $f(t) = \exp(t)$.

Range and nullspace

3.9 *Diagonal matrices.* A square matrix A is called diagonal if its off-diagonal elements are zero: $a_{ij} = 0$ for $i \neq j$.

(a) What are the range and the nullspace of the diagonal matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}?$$

(b) When is a diagonal matrix invertible?

3.10 Do the following matrices A have a zero nullspace?

(a) $A = \begin{bmatrix} -1 & 2 \\ 3 & -6 \\ 2 & -1 \end{bmatrix}$

(b) $A = \begin{bmatrix} -1 & 3 & 2 \\ 2 & -6 & -1 \end{bmatrix}$ (*i.e.*, the transpose of the matrix in part (a))

(c) $A = \begin{bmatrix} D \\ B \end{bmatrix}$, where B is $m \times n$ and D is a diagonal $n \times n$-matrix with nonzero diagonal elements.

(d) $A = I - U$ where U is $n \times n$ with $\|U\| < 1$.

Nonsingular matrices

3.11 Suppose A is a nonsingular matrix of order n, u and v are n-vectors, and $v^T A^{-1} u \neq -1$. Show that $A + uv^T$ is nonsingular with inverse

$$(A + uv^T)^{-1} = A^{-1} - \frac{1}{1 + v^T A^{-1} u} A^{-1} u v^T A^{-1}.$$

3.12 Suppose A is a nonsingular matrix of order n. Consider the matrix $2n \times 2n$-matrix M defined as

$$M = \begin{bmatrix} A & A + A^{-T} \\ A & A \end{bmatrix}.$$

(a) Show that M is nonsingular, by showing that $Mx = 0$ implies $x = 0$.

(b) Find the inverse of M. To find M^{-1}, express it as a block matrix

$$M^{-1} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}$$

with blocks of dimension $n \times n$, and determine the matrices W, X, Y, Z from the condition $MM^{-1} = I$.

3.13 *Inverse of a block-triangular matrix.* Express the inverse of the 2×2 block matrix

$$A = \begin{bmatrix} B & 0 \\ C & D \end{bmatrix}$$

in terms of B, C, D, B^{-1} and D^{-1}. We assume B is $n \times n$, C is $p \times n$, and D is $p \times p$, with B and D nonsingular.

3.14 *Lagrange interpolation.* We have already seen one method for finding a polynomial

$$p(t) = x_1 + x_2 t + \cdots + x_n t^{n-1} \tag{3.22}$$

with specified values

$$p(t_1) = y_1, \qquad p(t_2) = y_2, \qquad \ldots, \qquad p(t_n) = y_n. \tag{3.23}$$

The polynomial p is called the *interpolating* polynomial through the points (t_1, y_1), $\ldots$, (t_n, y_n). Its coefficients can be computed by solving the set of linear equations

$$\begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-2} & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-2} & t_2^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & t_{n-1} & \cdots & t_{n-1}^{n-2} & t_{n-1}^{n-1} \\ 1 & t_n & \cdots & t_n^{n-2} & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}. \tag{3.24}$$

The coefficient matrix is called a *Vandermonde* matrix. As we have seen in the lecture, a Vandermonde matrix is nonsingular if the points t_i are distinct ($t_i \neq t_j$ for $i \neq j$). As a consequence, the interpolating polynomial is unique: if the points t_i are distinct, then there exists exactly one polynomial of degree less than or equal to $n-1$ that satisfies (3.23), and its coefficients are the solution of the equations (3.24).

In this problem we describe another method for finding p, known as *Lagrange interpolation*.

(a) We define n polynomials l_i:

$$l_i(t) = \frac{\prod_{j \neq i}(t - t_j)}{\prod_{j \neq i}(t_i - t_j)}, \quad i = 1, \ldots, n.$$

Verify that l_i is a polynomial of degree $n - 1$, and that

$$l_i(t_k) = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{if } k \neq i. \end{cases}$$

For example, for $n = 3$, we have the three polynomials

$$l_1(t) = \frac{(t - t_2)(t - t_3)}{(t_1 - t_2)(t_1 - t_3)}, \quad l_2(t) = \frac{(t - t_1)(t - t_3)}{(t_2 - t_1)(t_2 - t_3)}, \quad l_3(t) = \frac{(t - t_1)(t - t_2)}{(t_3 - t_1)(t_3 - t_2)}.$$

(b) Show that the polynomial

$$p(t) = y_1 l_1(t) + y_2 l_2(t) + \cdots + y_n l_n(t) \tag{3.25}$$

has degree $n-1$ or less, and satisfies the interpolation conditions (3.23). It is therefore equal to the unique interpolating polynomial through those points.

(c) This provides another method for polynomial interpolation. To find the coefficients x_i we express the polynomial (3.25) in the form (3.22) by expanding the polynomials l_i as weighted sums of powers of t.

As an example, for $n = 3$, the polynomial (3.25) is given by

$$p(t) = y_1 \frac{(t-t_2)(t-t_3)}{(t_1-t_2)(t_1-t_3)} + y_2 \frac{(t-t_1)(t-t_3)}{(t_2-t_1)(t_2-t_3)} + y_3 \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)}.$$

Express this as $p(t) = x_1 + x_2 t + x_3 t^2$, *i.e.*, give expressions for x_1, x_2, x_3 in terms of t_1, t_2, t_3, y_1, y_2, y_3. Use the result to prove the following expression for the inverse of a 3×3 Vandermonde matrix:

$$\begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \end{bmatrix}^{-1} = \begin{bmatrix} \dfrac{t_2 t_3}{(t_1-t_2)(t_1-t_3)} & \dfrac{t_1 t_3}{(t_2-t_1)(t_2-t_3)} & \dfrac{t_1 t_2}{(t_3-t_1)(t_3-t_2)} \\ \dfrac{-t_2-t_3}{(t_1-t_2)(t_1-t_3)} & \dfrac{-t_1-t_3}{(t_2-t_1)(t_2-t_3)} & \dfrac{-t_1-t_2}{(t_3-t_1)(t_3-t_2)} \\ \dfrac{1}{(t_1-t_2)(t_1-t_3)} & \dfrac{1}{(t_2-t_1)(t_2-t_3)} & \dfrac{1}{(t_3-t_1)(t_3-t_2)} \end{bmatrix}.$$

Positive definite matrices

3.15 Are the following matrices positive definite?

(a) $A = \begin{bmatrix} -1 & 2 & 3 \\ 2 & 5 & -3 \\ 3 & -3 & 2 \end{bmatrix}$.

(b) $A = I - uu^T$ where u is an n-vector with $\|u\| < 1$.

(c) $A = \begin{bmatrix} I & B \\ B^T & I + B^T B \end{bmatrix}$ where B is an $m \times n$-matrix.

(d) $A = \begin{bmatrix} 1 & u^T \\ u & I \end{bmatrix}$ where u is an n-vector with $\|u\| < 1$.

3.16 Show that the inverse of a positive definite matrix is positive definite.

3.17 A square matrix P is called a *symmetric projection matrix* if $P = P^T$ and $P^2 = P$. Show that a symmetric projection matrix P satisfies the following properties.

(a) $I - P$ is also a symmetric projection matrix.

(b) $\|x\|^2 = \|Px\|^2 + \|(I-P)x\|^2$ for all x.

(c) P is positive semidefinite.

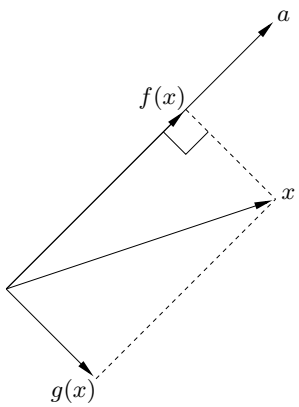
(d) If $P \neq 0$, then $\|P\| = 1$.

Carefully explain each step in your arguments.

3.18 Let a be a nonzero n-vector with $n \geq 2$. We define two $n \times n$-matrices

$$A = \frac{1}{\|a\|^2} aa^T, \qquad B = I - \frac{1}{\|a\|^2} aa^T.$$

The mapping $f(x) = Ax$ is the orthogonal projection of x on a. The mapping $g(x) = Bx = x - Ax$ is the difference between x and its projection on a.



(a) What are the norms of A and B?

(b) Are A and B nonsingular?

(c) Are A and B positive semidefinite?

Explain your answers.

3.19 Let A be an $m \times n$ matrix with $\|A\| < 1$.

(a) Show that the matrix $I - A^T A$ is positive definite.

(b) Show that the matrix

$$\begin{bmatrix} I & A \\ A^T & I \end{bmatrix}$$

is positive definite.

3.20 Suppose the matrix

$$\begin{bmatrix} 0 & A \\ A^T & B \end{bmatrix}$$

is positive semidefinite. Show that $A = 0$ and B is positive semidefinite.

Orthogonal matrices

3.21 A matrix of the form

$$P = I - \frac{2}{u^T u} u u^T,$$

where u is a nonzero n-vector, is called a *Householder* matrix of order n.

(a) Show that P is orthogonal.

(b) Is P positive definite?

3.22 A square matrix A is called *normal* if $AA^T = A^T A$. Show that if A is normal and nonsingular, then the matrix $Q = A^{-1}A^T$ is orthogonal.

3.23 Let S be a square matrix that satisfies $S^T = -S$. (This is called a *skew-symmetric* matrix.)

(a) Show that $I - S$ is nonsingular. (Hint: first show that $x^T S x = 0$ for all x.)

(b) Show that $(I+S)(I-S)^{-1} = (I-S)^{-1}(I+S)$. (This property does not rely on the skew-symmetric property; it is true for any matrix S for which $I-S$ is nonsingular.)

(c) Show that the matrix

$$A = (I + S)(I - S)^{-1}$$

is orthogonal.

Part II

Matrix algorithms

Chapter 4

Complexity of matrix algorithms

In the rest of the course we will study algorithms for various classes of mathematical problems (sets of linear equations, nonlinear equations, least-squares problems, et cetera). Different methods for the same problem can be compared on the basis of their computational cost or complexity, *i.e.*, the amount of work they involve or the time they require to find the solution. In this chapter, we describe a popular method for estimating the complexity of *non-iterative* matrix algorithms of the type discussed in chapters 5–10. The complexity of iterative methods will be discussed in chapter 11.

4.1 Flop counts

The cost of a matrix algorithm is often expressed by giving the total number of *floating-point operations* or *flops* required to carry it out, as a function of the 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

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

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 them 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 $(1/3)n^3$.

4.2 Vector operations

To compute the inner product $x^T y$ of two n-vectors x, y 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 if $n \gg 1$, or even more approximately, order n flops. A scalar vector multiplication αx, where α is a scalar and x is an n-vector costs n flops. The addition $x + y$ of two n-vectors x, y 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 elements, then the inner product $x^T y$ can be computed in $2N$ flops.

4.3 Matrix-vector multiplication

A matrix-vector multiplication $y = Ax$ where A is $m \times n$ costs $m(2n-1)$ flops, *i.e.*, $2mn$ if $n \gg 1$: we have to calculate m elements of y, each of which is the product of a row of A with x, *i.e.*, an inner product of n-vectors.

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 is represented (stored) in the factored form $A = UV$, where U is an $m \times p$ matrix and V is a $p \times n$ matrix. 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. If $p \ll \min\{m, n\}$, this is small compared to $2mn$.

4.4 Matrix-matrix multiplication

The matrix-matrix product $C = AB$, where A is an $m \times n$ matrix and B is $n \times p$, costs $mp(2n-1)$ flops, or $2mnp$ if $n \gg 1$. 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 is $m \times n$, B is $n \times p$, and C is $p \times q$. 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

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

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.

Example The following MATLAB code generates two $n \times n$-matrices A and B and an n-vector x. It then compares the CPU time required to evaluate the product $y = ABx$ using two methods. In the first method we first compute the matrix AB and then multiply this matrix with x. In the second method we first compute the vector Bx and then multiply this vector with A.

```
A = randn(n,n);  B = randn(n,n);  x = randn(n,1);
t1 = cputime;  y = (A*B)*x;  t1 = cputime - t1;
t2 = cputime;  y = A*(B*x);  t2 = cputime - t2;
```

The table shows the results on a 2.8GHz machine for eight values of n.

n	time (sec.) method 1	time (sec.) method 2
500	0.10	0.004
1000	0.63	0.02
1500	1.99	0.06
2000	4.56	0.11
2500	8.72	0.17
3000	14.7	0.25
4000	34.2	0.46
5000	65.3	0.68

Let us first derive the flop counts for the two methods. In method 1 we first compute AB, which costs $2n^3$ flops, and then multiply the matrix AB with a vector, which costs $2n^2$ flops. The total cost is $2n^3 + 2n^2$, or $2n^3$ if we only keep the leading term. In method 2, we make two matrix-vector products of size n, and each costs $2n^2$ flops. The total cost is therefore $4n^2$.

Comparing the flop counts with the CPU times in the table, we can make several observations. First, the simple flop count analysis leads us to the correct conclusion that for large n, method 2 is a lot faster. The flop count predicts that if we double n, the cost of method 1 goes up by a factor of 8, while the cost of method 2 goes up by 4. We see that this is roughly the case for the larger values of n.

A closer look also shows the limitations of the flop count analysis. For example, for $n = 5000$, method 1 is about 96 times slower than method 2, although the flop count would predict a factor of $2n^3/(4n^2) = 2500$.

Exercises

4.1 Give the number of flops required to evaluate a product of three matrices

$$X = ABC,$$

where A is $n \times n$, B is $n \times 10$, and C is $10 \times n$. You can evaluate the product in two possible ways:

(a) from left to right, as $X = (AB)C$

(b) from right to left, as $X = A(BC)$

Which method is faster for large n? (You should assume that A, B, and C are dense, *i.e.*, they do not possess any special structure that you can take advantage of.)

4.2 A matrix C is defined as

$$C = Auv^T B$$

where A and B are $n \times n$-matrices, and u and v are n-vectors. The product on the right-hand side can be evaluated in many different ways, *e.g.*, as $A(u(v^T B))$ or as $A((uv^T)B)$, etc. What is the fastest method (requiring the least number of flops) when n is large?

4.3 The following MATLAB code generates three random vectors u, v, x of dimension $n = 1000$, and calculates the CPU time required to evaluate $y = (I + uv^T)x$ using two equivalent expressions.

```
n = 1000;
u = randn(n,1);  v = randn(n,1);  x = randn(n,1);
t1 = cputime;  y = (eye(n)+u*v')*x;  t1 = cputime - t1
t2 = cputime;  y = x + (v'*x)*u;  t2 = cputime - t2
```

Run this code, and compare t_1 and t_2. Repeat for $n = 2000$. Are your observations consistent with what you expect based on a flop count?

Chapter 5

Triangular matrices

This chapter is the first of three chapters that deal with the solution of square sets of linear equations

$$Ax = b.$$

Here we discuss the special case of triangular A. We show that a triangular matrix is nonsingular if it has nonzero diagonal elements, and that a triangular set of n equations in n variables can be solved in n^2 flops.

5.1 Definitions

A matrix A is *lower triangular* if the elements above the diagonal are zero, *i.e.*, $a_{ij} = 0$ for $i < j$. The matrix

$$\begin{bmatrix} -2 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -7 & 3 & 0 \\ -2 & 8 & 6 & -5 \end{bmatrix}$$

is an example of a 4×4 lower triangular matrix. Note that the elements on or below the diagonal do not have to be nonzero (in this example, the $3, 1$ element is zero).

A matrix A is *upper triangular* if the elements below the diagonal are zero: $a_{ij} = 0$ for $i > j$. Equivalently, a matrix is upper triangular if its transpose is lower triangular.

A lower triangular matrix is *unit lower triangular* if the diagonal elements are equal to one. An upper triangular matrix is *unit upper triangular* if the diagonal elements are equal to one.

A matrix is *diagonal* if the off-diagonal elements are zero: $a_{ij} = 0$ for $i \neq j$. A diagonal matrix is both upper triangular and lower triangular.

5.2 Forward substitution

Suppose A is a lower triangular matrix of order n with nonzero diagonal elements. (We will return to the case where one or more diagonal elements are zero.) Consider the equation $Ax = b$:

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

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 right-hand side is known.) Continuing this way, we can express each element of x in terms of previous elements, 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 algorithm is called *forward substitution*, since we successively compute the elements of x by substituting the known values into the next equation.

Flop count 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 with nonzero diagonal elements, we can solve $Ax = b$ 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 diagonal with nonzero diagonal elements, then the algorithm reduces to

$$x_1 := b_1/a_{11}, \qquad x_2 := b_2/a_{22}, \qquad \ldots, \qquad x_n := b_n/a_{nn},$$

and the cost is n flops. If A is sparse, with at most k off-diagonal nonzero elements 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 $2n$.

5.3 Back substitution

If A is an upper triangular matrix with nonzero diagonal elements, we can solve $Ax = b$ 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 elements in backwards order. The cost of backward substitution is n^2 flops if A is $n \times n$.

5.4 Inverse of a triangular matrix

Nonsingular triangular matrix If A is (upper or lower) triangular with nonzero diagonal elements, the equation $Ax = b$ is solvable (by forward or backward substitution) for every value of b. This shows that a triangular matrix with nonzero diagonal elements has a full range. It therefore satisfies the ten properties on page 61. In particular, it is nonsingular and invertible.

Inverse To compute the inverse of a nonsingular triangular matrix we can solve the matrix equation $AX = I$ column by column. The equation can be written as

$$AX_1 = e_1, \qquad AX_2 = e_2, \qquad \ldots, \qquad AX_n = e_n, \tag{5.1}$$

where the n-vectors X_k are the n columns of X and the vectors e_k are the unit vectors of size n. Each of these equations has a unique solution X_k, and can be solved by forward substitution (if A is lower triangular) or back substitution (if A is upper triangular).

The inverse of the matrix on page 79, for example, is

$$\begin{bmatrix} -1/2 & 0 & 0 & 0 \\ -1/2 & -1 & 0 & 0 \\ -7/6 & -7/3 & 1/3 & 0 \\ -2 & -22/5 & 2/5 & -1/5 \end{bmatrix}.$$

The inverse of a lower triangular matrix is lower triangular and the inverse of an upper triangular matrix is upper triangular. For example, if A is lower triangular and we solve $AX_k = e_k$ in (5.1) by forward substitution, then we find that the first $k-1$ elements of the kth column X_k are zero.

Singular triangular matrices For completeness, we examine the case of a triangular matrix with one or more zero elements on the diagonal. Such a matrix is singular, as can be seen in many different ways, for example, by showing that the matrix has a nonzero nullspace.

If A is lower triangular with $a_{nn} = 0$, then $x = (0, \ldots, 0, 1)$ is in its nullspace. If $a_{nn} \neq 0$ but there are zero elements on the diagonal, then A can be partitioned as

$$A = \begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & 0 & 0 \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

where the zero on the diagonal is a scalar, the blocks A_{11} and A_{33} are square and lower triangular, and the diagonal elements of A_{33} are nonzero, *i.e.*, A_{33} is nonsingular (whereas A_{11} may have zeros on the diagonal). Then one can verify that A has nonzero vectors in its nullspace. For example,

$$\begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & 0 & 0 \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -A_{33}^{-1} A_{32} \end{bmatrix} = 0.$$

Exercises

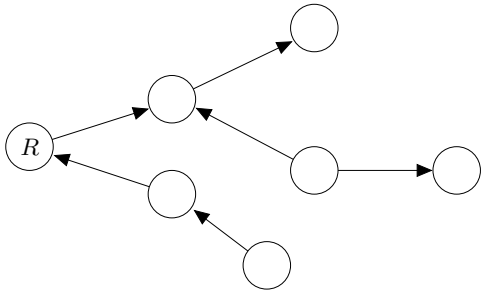
5.1 Let A be a lower triangular matrix of order n. Verify the following properties.

(a) If B is lower triangular of order n, then the product AB is lower triangular.

(b) The matrix A^k is lower triangular for all positive integers k.

(c) If A is nonsingular, then A^k is lower triangular for all integers k (positive or negative).

5.2 A *directed tree* is a connected graph with $n+1$ nodes and n directed arcs. The figure shows an example with $n = 6$ (7 nodes and 6 arcs).



A directed tree can be represented by a *reduced node-arc incidence matrix* defined as follows. We label one node as the *root* node. The other nodes are labeled with integers from 1 to n. The arcs are also given integer labels from 1 to n. The node-arc incidence matrix A is then defined as the $n \times n$ matrix with elements

$$A_{ij} = \begin{cases} +1 & \text{if arc } j \text{ ends at node } i \\ -1 & \text{if arc } j \text{ begins at node } i \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, n$.

(a) Suppose we use the node labeled R as the root node in the example. Number the non-root nodes and the arcs in such a way that the node-incidence matrix is lower triangular.

(b) Show that the node-incidence matrix obtained in part (a) is nonsingular.

(c) Show that the elements of the inverse of the node-incidence matrix in part (a) have values 0, +1 or −1.

5.3 Let A be a nonsingular triangular matrix of order n.

(a) What is the cost of computing A^{-1}?

(b) What is the cost of solving $Ax = b$ by first computing A^{-1} and then forming the matrix-vector product $x = A^{-1}b$? Compare with the cost of forward and backward substitution.

5.4 A lower triangular matrix A is called a lower triangular band matrix with k subdiagonals if $A_{ij} = 0$ for $i > j + k$. The matrix

$$A = \begin{bmatrix} -0.9 & 0 & 0 & 0 & 0 & 0 \\ 0.7 & -0.7 & 0 & 0 & 0 & 0 \\ 1.4 & -2.7 & 3.7 & 0 & 0 & 0 \\ 0 & 0.6 & 0.3 & -1.2 & 0 & 0 \\ 0 & 0 & -2.2 & 1.1 & -0.6 & 0 \\ 0 & 0 & 0 & 2.4 & 2.4 & 0.7 \end{bmatrix}$$

is a 6×6 lower triangular band matrix with 2 subdiagonals.

What is the cost of solving a set of linear equations $Ax = b$ if A is an $n \times n$ lower triangular band matrix with k subdiagonals and nonzero diagonal elements? Express the cost as the number of flops as a function of n and k. You only have to give the dominant term in the flop count and you can assume that $k \ll n$.

5.5 Describe an efficient method for each of the following two problems, and give a complexity estimate (number of flops for large n).

(a) Solve

$$DX + XD = B$$

where D is $n \times n$ and diagonal. The diagonal elements of D satisfy $d_{ii} + d_{jj} \neq 0$ for all i and j. The matrices D and B are given. The variable is the $n \times n$-matrix X.

(b) Solve

$$LX + XL^T = B$$

where L is lower triangular. The diagonal elements of L satisfy $l_{ii} + l_{jj} \neq 0$ for all i and j. The matrices L and B are given. The variable is the $n \times n$-matrix X. (*Hint:* Solve for X column by column.)

If you know several methods, choose the fastest one (least number of flops for large n).

Chapter 6

Cholesky factorization

The next class of problems are sets of linear equations $Ax = b$ with positive definite A.

6.1 Definition

Every positive definite matrix A can be factored as

$$A = LL^T$$

where L is lower triangular with positive diagonal elements. This is called the *Cholesky factorization* of A. The matrix L, which is uniquely determined by A, is called the *Cholesky factor* of A. If $n = 1$ (*i.e.*, A is a positive scalar), then L is just the square root of A. So we can also interpret the Cholesky factor as the 'square root' of a positive definite matrix A.

An example of a 3×3 Cholesky factorization is

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix}. \tag{6.1}$$

In section 6.4 we will discuss how the Cholesky factorization is computed, and we will find that the cost of factoring a matrix of order n is $(1/3)n^3$ flops. But let us first explain why the Cholesky factorization is useful.

6.2 Positive definite sets of linear equations

It is straightforward to solve $Ax = b$, with A positive definite, if we first compute the Cholesky factorization $A = LL^T$. To solve $LL^T x = b$, we introduce an intermediate

variable $w = L^T x$, and compute w and x by solving two triangular sets of equations. We first solve

$$Lw = b$$

for w. Then we solve

$$L^T x = w$$

to find x. These two equations are solvable, because L is lower triangular with positive diagonal elements (see last chapter).

Algorithm 6.1. Solving linear equations by Cholesky factorization.

given a set of linear equations with A positive definite of order n.

1. *Cholesky factorization:* factor A as $A = LL^T$ ($(1/3)n^3$ flops).
2. *Forward substitution:* solve $Lw = b$ (n^2 flops).
3. *Back substitution:* solve $L^T x = w$ (n^2 flops).

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

Example We compute the solution of

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 30 \\ 15 \\ -16 \end{bmatrix},$$

using the factorization in (6.1). We first solve

$$\begin{bmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 30 \\ 15 \\ -16 \end{bmatrix}$$

by forward substitution. The solution is $w = (6, -1, -3)$. Then we solve

$$\begin{bmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -1 \\ -3 \end{bmatrix}$$

by back substitution and obtain the solution $x = (1, 0, -1)$.

Equations with multiple right-hand sides Suppose we need to solve m equations

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

where A is positive definite of order n, and $b_1, \ldots, b_m$ are n-vectors. These m sets of equations have the same coefficient matrix but different right-hand sides $b_1, \ldots, b_m$. Alternatively, we can think of the problem as solving the matrix equation

$$AX = B$$

where X and B are the $n \times m$-matrices

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix}, \qquad B = \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix}.$$

We first compute the Cholesky factorization of A, which costs $(1/3)n^3$. Then for $i = 1, \ldots, m$, we solve $Ax_k = b_k$ using forward and backward substitution (this costs $2n^2$ flops per right-hand side). Since we only factor A once, the total effort is

$$(1/3)n^3 + 2mn^2$$

flops. Had we (needlessly) repeated the factorization step for each right-hand side, the cost would have been $m((1/3)n^3 + 2n^2)$. Since the factorization cost $((1/3)n^3)$ dominates the cost of forward and backward substitution $(2n^2)$, this method allows us to solve a small number of linear systems, with the same coefficient matrix, at roughly the cost of solving one.

6.3 Inverse of a positive definite matrix

If A is positive definite with Cholesky factorization $A = LL^T$, then its inverse can be expressed as

$$A^{-1} = L^{-T}L^{-1}.$$

This follows from the expressions for the inverse of a product of nonsingular matrices and the inverse of a transpose (see section 3.3.3) and the fact that L is invertible because it is lower-triangular with nonzero diagonal elements.

This expression provides another interpretation of algorithm 6.1. If we multiply both sides of the equation $Ax = b$ on the left by A^{-1}, we get

$$x = A^{-1}b = L^{-T}L^{-1}b.$$

Steps 2 and 3 of the algorithm evaluate this expression by first computing $w = L^{-1}b$, and then $x = L^{-T}w = L^{-T}L^{-1}b$.

Computing the inverse The inverse of a positive definite matrix can be computed by solving

$$AX = I,$$

using the method for solving equations with multiple right-hand sides described in the previous section. Only one Cholesky factorization of A is required, and n forward and backward substitutions. The cost of computing the inverse using this method is $(1/3)n^3 + 2n^3 = (7/3)n^3$, or about $2n^3$.

6.4 Computing the Cholesky factorization

Algorithm 6.2 below is a recursive method for computing the Cholesky factorization of a positive definite matrix A of order n. It refers to the following block matrix

partitioning of A and its Cholesky factor L:

$$A = \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}, \qquad L = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$$

where a_{11} and l_{11} are scalars. By definition the Cholesky factor L is lower triangular with positive diagonal elements, so $l_{11} > 0$ and L_{22} is lower triangular with positive diagonal elements.

Algorithm 6.2. CHOLESKY FACTORIZATION.

given a positive definite matrix A of order n.

1. Calculate the first column of L: $l_{11} = \sqrt{a_{11}}$ and $L_{21} = (1/l_{11})A_{21}$.
2. Compute the Cholesky factorization $A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T$.

It can be shown that the cost of this algorithm is $(1/3)n^3$ flops.

To verify the correctness of the algorithm, we write out the equality $A = LL^T$ for the partitioned matrices:

$$\begin{aligned} \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} &= \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} \\ &= \begin{bmatrix} l_{11}^2 & l_{11}L_{21}^T \\ l_{11}L_{21} & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix}. \end{aligned}$$

The equality allows us to determine the first column of L:

$$l_{11} = \sqrt{a_{11}}, \qquad L_{21} = (1/l_{11})A_{21}.$$

(Recall that $a_{11} > 0$ if A is positive definite.) Furthermore,

$$L_{22}L_{22}^T = A_{22} - L_{21}L_{21}^T,$$

so we must choose L_{22} so that $L_{22}L_{22}^T$ is the Cholesky factorization of the matrix

$$A_{22} - L_{21}L_{21}^T = A_{22} - (1/a_{11})A_{21}A_{21}^T.$$

This is the Schur complement of a_{11} and, as we have seen in section 3.4.2, it is a positive definite matrix of order $n - 1$.

If we continue recursively, we arrive at a Cholesky factorization of a 1×1 positive definite matrix, which is trivial (it is just the square root). Algorithm 6.2 therefore works for all positive definite matrices A. It also provides a practical way of checking whether a given matrix is positive definite. If we try to factor a matrix that is not positive definite, we will encounter a nonpositive diagonal element a_{11} at some point during the recursion.

Example As an example, we derive the factorization (6.1).

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}.$$

We start with the first column of L:

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & l_{22} & 0 \\ -1 & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 5 & 3 & -1 \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}.$$

The remainder of the factorization follows from

$$\begin{aligned} \begin{bmatrix} 18 & 0 \\ 0 & 11 \end{bmatrix} &= \begin{bmatrix} 3 & l_{22} & 0 \\ -1 & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 3 & -1 \\ l_{22} & l_{32} \\ 0 & l_{33} \end{bmatrix} \\ &= \begin{bmatrix} 3 \\ -1 \end{bmatrix} \begin{bmatrix} 3 & -1 \end{bmatrix} + \begin{bmatrix} l_{22} & 0 \\ l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{22} & l_{32} \\ 0 & l_{33} \end{bmatrix}, \\ \begin{bmatrix} 9 & 3 \\ 3 & 10 \end{bmatrix} &= \begin{bmatrix} l_{22} & 0 \\ l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{22} & l_{32} \\ 0 & l_{33} \end{bmatrix}. \end{aligned}$$

To factor this 2×2-matrix, we first determine l_{22} and l_{32}:

$$\begin{bmatrix} 9 & 3 \\ 3 & 10 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 1 & l_{33} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & l_{33} \end{bmatrix}.$$

Finally, the last element follows from

$$10 = 1 + l_{33}^2$$

i.e., $l_{33} = 3$.

6.5 Sparse positive definite matrices

If A is very sparse (*i.e.*, most of its elements are zero), then its Cholesky factor L is usually quite sparse as well, and the Cholesky factorization can be computed in much less than $(1/3)n^3$ flops (provided we store A in a format that allows us to skip multiplications and additions with zero). If L is sparse, the associated forward and backward substitutions can also be carried out in much less than $2n^2$ flops.

The sparsity pattern of L (*i.e.*, the location of its zeros and nonzeros) can be determined from the sparsity pattern of A. Let us examine the first two steps of algorithm 6.2. The sparsity pattern of L_{21}, computed in the first step, is exactly the same as the sparsity pattern of A_{21}. In step 2, the matrix $A_{22} - L_{21}L_{21}^T$ is formed, and this matrix will usually have some nonzero elements in positions where A_{22} is zero. More precisely, the i, j element of $A_{22} - L_{21}L_{21}^T$ can be nonzero if the i, j element of A_{22} is nonzero, but also if the ith and the jth elements of L_{21} are

nonzero. If we then proceed to the next cycle in the recursion and determine the first column of the Cholesky factor of $A_{22} - L_{21}L_{21}^T$, some nonzero elements may be introduced in positions of L where the original matrix A had a zero element. The creation of new nonzero elements in the sparsity pattern of L, compared to the sparsity pattern of A, is called *fill-in*. If the amount of fill-in is small, L will be sparse, and the Cholesky factorization can be computed very efficiently.

In some cases the fill-in is very extensive, or even complete. Consider for example, the set of linear equations

$$\begin{bmatrix} 1 & a^T \\ a & I \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix} \tag{6.2}$$

where a is an n-vector with $\|a\| < 1$. It can be shown that the coefficient matrix is positive definite (see exercise 3.15). Its Cholesky factorization is

$$\begin{bmatrix} 1 & a^T \\ a & I \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a & L_{22} \end{bmatrix} \begin{bmatrix} 1 & a^T \\ 0 & L_{22}^T \end{bmatrix}$$

where $I - aa^T = L_{22}L_{22}^T$ is the factorization of $I - aa^T$. Now if all the elements of a are nonzero, then $I - aa^T$ is a dense matrix. Therefore L_{22} will be dense and we have 100% fill-in.

If, on the other hand, we first rewrite the equations (6.2) as

$$\begin{bmatrix} I & a \\ a^T & 1 \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} c \\ b \end{bmatrix}$$

(by reordering the variables and right-hand sides) and then factor the reordered coefficient matrix, we obtain a factorization with zero fill-in:

$$\begin{bmatrix} I & a \\ a^T & 1 \end{bmatrix} = \begin{bmatrix} I & 0 \\ a^T & \sqrt{1 - a^T a} \end{bmatrix} \begin{bmatrix} I & a \\ 0 & \sqrt{1 - a^T a} \end{bmatrix}.$$

This simple example shows that reordering the equations and variables can have a dramatic effect on the amount of fill-in, and hence on the efficiency of solving a sparse set of equations. To describe this more generally, we introduce permutation matrices.

Permutation matrices Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation of $(1, 2, \ldots, n)$. The associated *permutation matrix* is the $n \times n$ matrix P with elements

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

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

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

For example, the permutation matrix associated with $\pi = (2, 4, 1, 3)$ is

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and $Px = (x_2, x_4, x_1, x_3)$.

The elements of the transpose of a permutation matrix are

$$(P^T)_{ij} = \begin{cases} 1 & i = \pi_j \\ 0 & \text{otherwise.} \end{cases}$$

This is also a permutation matrix, and the corresponding permutation is the inverse permutation π^{-1}. (The inverse permutation is defined by $(\pi^{-1})_i = j$ where $i = \pi_j$.) The inverse of the permutation $\pi = (2, 4, 1, 3)$ in the example is $\pi^{-1} = (3, 1, 4, 2)$, and the corresponding permutation matrix is

$$P^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

If P is a permutation matrix, then $P^T P = PP^T = I$. (In other words, $P^T = P^{-1}$.) This makes solving $Px = b$ extremely easy. The solution is $x = P^T b$, and 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). For example, the solution of the equation

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

is $x = (b_3, b_1, b_4, b_2)$.

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

$$A = PLL^T P^T \tag{6.3}$$

where P is a permutation matrix and L is lower triangular with positive diagonal elements. We can express this as $P^T AP = LL^T$, *i.e.*, LL^T is the Cholesky factorization of $P^T AP$, the matrix A with its rows and columns permuted by the same permutation.

The matrix $P^T AP$ arises when we reorder the equations and the variables in $Ax = b$. If we define new variables $\tilde{x} = P^T x$ and apply the same permutation to the right-hand side to get $\tilde{b} = P^T b$, then $\tilde{x}$ satisfies the equation $(P^T AP)\tilde{x} = \tilde{b}$.

Since $P^T AP$ is positive definite for any permutation matrix P, we are free to choose any permutation matrix; for every P there is a unique Cholesky factor L. The choice of P, however, can greatly affect the sparsity of L, which in turn can greatly affect the efficiency of solving $Ax = b$. Very effective heuristic methods are known to select a permutation matrix P that leads to a sparse factor L.

Exercises

6.1 Compute the Cholesky factorization of

$$A = \begin{bmatrix} 4 & 6 & 2 & -6 \\ 6 & 34 & 3 & -9 \\ 2 & 3 & 2 & -1 \\ -6 & -9 & -1 & 38 \end{bmatrix}$$

You can use MATLAB to verify the result, but you have to provide the details of your calculations.

6.2 For what values of a are the following matrices positive definite? To derive the conditions, factor A using a Cholesky factorization and collect the conditions on a needed for the factorization to exist.

(a) $A = \begin{bmatrix} 1 & a \\ a & 1 \end{bmatrix}$.

(b) $A = \begin{bmatrix} 1 & a & 0 \\ a & 1 & a \\ 0 & a & 1 \end{bmatrix}$.

(c) $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & a \end{bmatrix}$.

(d) $A = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ a & 0 & 1 \end{bmatrix}$.

(e) $A = \begin{bmatrix} a & 1 & 0 \\ 1 & -a & 1 \\ 0 & 1 & a \end{bmatrix}$.

(f) $A = \begin{bmatrix} I & aI \\ aI & I \end{bmatrix}$. I is the $n \times n$ identity matrix.

(g) $A = \begin{bmatrix} I & au \\ au^T & 1 \end{bmatrix}$. I is the $n \times n$ identity matrix and $u = (1, 1, \ldots, 1)$, the n-vector with all its elements equal to one.

(h) $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & a & a \\ 1 & a & 2 \end{bmatrix}$.

6.3 Suppose A is an $n \times n$ positive definite matrix. For what values of the scalar β is the matrix

$$\begin{bmatrix} A & -A \\ -A & \beta A \end{bmatrix}$$

positive definite?

6.4 Let A be a positive definite matrix of size $n \times n$. For what values of the scalar a are the following matrices positive definite?

(a) $\begin{bmatrix} A & ae_1 \\ ae_1^T & 1 \end{bmatrix}$ (b) $\begin{bmatrix} A & e_1 \\ e_1^T & a \end{bmatrix}$ (c) $\begin{bmatrix} A & ae_1 \\ ae_1^T & a \end{bmatrix}$.

($e_1 = (1, 0, \ldots, 0)$ denotes the first unit vector of length n.) Give your answer for each of the three problems in the form of upper and/or lower bounds ($a_{\min} < a < a_{\max}$, $a > a_{\min}$, or $a < a_{\max}$). Explain how you can compute the limits $a_{\min}$ and $a_{\max}$ using the Cholesky factorization $A = LL^T$. (You don't need to discuss the complexity of the computation.)

6.5 Let A be defined as

$$A = I + BB^T$$

where B is a given orthogonal $n \times m$ matrix (not necessarily square).

(a) Show that A is positive definite.

(b) What is the cost (number of flops for large m and n) of solving $Ax = b$ by first computing $A = I + BB^T$ and then applying the standard method for linear equations $Ax = b$ with positive definite A?

(c) Show that $A^{-1} = I - (1/2)BB^T$.

(d) Use the expression in part (c) to derive an efficient method for solving $Ax = b$ (*i.e.*, a method that is much more efficient than the method in part (b).) Give the cost of your method (number of flops for large m and n).

6.6 You are given the Cholesky factorization $A = LL^T$ of a positive definite matrix A of order n, and an n-vector u.

(a) What is the Cholesky factorization of the $(n+1) \times (n+1)$-matrix

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

You can assume that B is positive definite.

(b) What is the cost of computing the Cholesky factorization of B, if the factorization of A (*i.e.*, the matrix L) is given?

(c) Suppose $\|L^{-1}\| \leq 1$. Show that B is positive definite for all u with $\|u\| < 1$.

6.7 A matrix A is *tridiagonal* if $a_{ij} = 0$ for $|i - j| > 1$, *i.e.*, A has the form

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & 0 \\ 0 & 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

What is the cost of computing the Cholesky factorization of a general tridiagonal positive definite matrix of size $n \times n$? Count square roots as one flop and keep only the leading term in the total number of flops. Explain your answer.

6.8 Define a block matrix

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

where the three matrices A, B, C have dimension $n \times n$. The matrices A and C are symmetric and positive definite. Show that K can be factored as

$$K = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix},$$

where L_{11} and L_{22} are lower triangular matrices with positive diagonal elements. The blocks L_{11}, L_{21}, L_{22}, and the two identity matrices on the right-hand side, all have size $n \times n$.

What is the cost of computing this factorization (number of flops for large n)? Carefully explain your answers.

6.9 Let u be a nonzero n-vector.

(a) Show that $I + auu^T$ is positive definite if $a > -1/\|u\|^2$.

(b) Suppose $a > -1/\|u\|^2$. It can be shown that the triangular factor in the Cholesky factorization $I + auu^T = LL^T$ can be written as

$$L = \begin{bmatrix} \beta_1 & 0 & 0 & \cdots & 0 & 0 \\ \gamma_1 u_2 & \beta_2 & 0 & \cdots & 0 & 0 \\ \gamma_1 u_3 & \gamma_2 u_3 & \beta_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma_1 u_{n-1} & \gamma_2 u_{n-1} & \gamma_3 u_{n-1} & \cdots & \beta_{n-1} & 0 \\ \gamma_1 u_n & \gamma_2 u_n & \gamma_3 u_n & \cdots & \gamma_{n-1} u_n & \beta_n \end{bmatrix}.$$

The entries below the diagonal in column k are the entries $u_{k+1}, \ldots, u_n$ of the vector u, multiplied with γ_k.

Give an algorithm, with a complexity linear in n, for calculating the parameters $\beta_1, \ldots, \beta_n$ and $\gamma_1, \ldots, \gamma_{n-1}$, given the vector u and the scalar a.

6.10 Let A be a positive definite 5×5-matrix with the nonzero pattern

$$A = \begin{bmatrix} \bullet & & \bullet & \bullet & \\ & \bullet & & \bullet & \\ \bullet & & \bullet & & \bullet \\ \bullet & \bullet & & \bullet & \\ & & \bullet & & \bullet \end{bmatrix}.$$

The dots indicate the positions of the nonzero elements; all the other elements are zero.

The Cholesky factor L of A has one of the following lower-triangular nonzero patterns. Which one is correct? For each L, explain why L is or is not the Cholesky factor of A.

(a) $L =$
$$\begin{bmatrix} \bullet & & & & \\ & \bullet & & & \\ \bullet & & \bullet & & \\ & \bullet & & \bullet & \\ \bullet & & \bullet & & \bullet \end{bmatrix}$$

(b) $L =$
$$\begin{bmatrix} \bullet & & & & \\ & \bullet & & & \\ \bullet & \bullet & \bullet & & \\ \bullet & \bullet & & \bullet & \\ & & \bullet & & \bullet \end{bmatrix}$$

(c) $L =$
$$\begin{bmatrix} \bullet & & & & \\ & \bullet & & & \\ \bullet & & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \\ & \bullet & & \bullet & \bullet \end{bmatrix}$$

(d) $L =$
$$\begin{bmatrix} \bullet & & & & \\ & \bullet & & & \\ \bullet & & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \\ & & \bullet & \bullet & \bullet \end{bmatrix}$$

Chapter 7

LU factorization

In this chapter we discuss the standard method for solving a square set of linear equations $Ax = b$ with nonsingular coefficient matrix A.

7.1 Factor-solve method

Algorithm 6.1 for solving positive definite linear equations divides the solution in two parts. First the matrix A is expressed as a product of two triangular matrices; then the solution x is computed by solving two triangular sets of equations. This idea extends to linear equations with general coefficient matrices and is called the *factor-solve* approach to solving $Ax = b$. In the factor-solve approach we first express the matrix A as a product of a small number of matrices with special properties:

$$A = A_1 A_2 \cdots A_k$$

(usually $k \leq 4$). This is the *factorization step*. We then solve the equation

$$A_1 A_2 \cdots A_k x = b$$

by introducing intermediate variables $z_1, \ldots, z_{k-1}$, and by solving k linear equations

$$A_1 z_1 = b, \qquad A_2 z_2 = z_1, \qquad \ldots, \qquad A_{k-1} z_{k-1} = z_{k-2}, \qquad A_k x = z_{k-1}. \quad (7.1)$$

This is called the *solve step*. If each of the equations in (7.1) is easy to solve (for example, because A_i is triangular or diagonal or a permutation matrix), this gives a method for solving $Ax = b$.

The method for sparse positive definite equations in section 6.5 is another illustration of the factor-solve method. Here A is factored as a product of four matrices $PLL^T P^T$. In the solve step we solve four simple equations

$$Pz_1 = b, \qquad Lz_2 = z_1, \qquad L^T z_3 = z_2, \qquad P^T x = z_3.$$

The total flop count for solving $Ax = b$ using the 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 the example of algorithm 6.1, we have $f = (1/3)n^3$ and $s = 2n^2$. In many applications, the cost of the factorization, f, dominates the solve cost s. If that is the case, the cost of solving $Ax = b$ is just f.

Equations with multiple right-hand sides When the factorization cost f dominates the solve cost s, the factor-solve method allows us to solve a small number of linear systems

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

with the same coefficient matrix but different right-hand sides, at essentially the same cost as solving one. Since we only factor A once, the total effort is

$$f + ms,$$

as opposed to $m(f + s)$ for solving m systems with different coefficient matrices.

7.2 Definition

Every square nonsingular $n \times n$-matrix A can be factored as

$$A = PLU$$

where P is an $n \times n$-permutation matrix, L is a unit lower triangular $n \times n$-matrix (a lower triangular matrix with diagonal elements equal to one), and U is a nonsingular upper triangular $n \times n$-matrix. This is called the *LU factorization* of A. We can also write the factorization as $P^T A = LU$, where the matrix $P^T A$ is obtained from A by reordering the rows.

An example of a 3×3 LU factorization is

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 9 & 0 \\ 6 & 8 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 0 & 15/19 & 1 \end{bmatrix} \begin{bmatrix} 6 & 8 & 8 \\ 0 & 19/3 & -8/3 \\ 0 & 0 & 135/19 \end{bmatrix}, \quad (7.2)$$

as can be verified by multiplying out the right-hand side. Another factorization of the same matrix is

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 9 & 0 \\ 6 & 8 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & -19/5 & 1 \end{bmatrix} \begin{bmatrix} 2 & 9 & 0 \\ 0 & 5 & 5 \\ 0 & 0 & 27 \end{bmatrix},$$

which shows that the LU factorization is not always unique.

The standard algorithm for computing an LU factorization is called *Gaussian elimination with partial pivoting* (GEPP) or *Gaussian elimination with row pivoting*, and will be described in section 7.5. The cost is $(2/3)n^3$ flops.

7.3 Nonsingular sets of linear equations

If we use the LU factorization in the factor-solve method we obtain an algorithm for solving linear equations with a general nonsingular coefficient matrix.

Algorithm 7.1. SOLVING LINEAR EQUATIONS BY LU FACTORIZATION.

given a set of linear equations $Ax = b$ with A $n \times n$ and nonsingular.

1. *LU factorization:* factor A as $A = PLU$ ($(2/3)n^3$ flops).
2. *Permutation:* $v = P^T b$ (0 flops).
3. *Forward substitution:* solve $Lw = v$ (n^2 flops).
4. *Back substitution:* solve $Ux = w$ (n^2 flops).

The cost of the factorization step is $(2/3)n^3$, and the cost of the three other steps is $2n^2$. The total cost is $(2/3)n^3 + 2n^2$, or $(2/3)n^3$ flops if we keep only the leading term. This algorithm is the standard method for solving linear equations.

Example We solve the equations

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 9 & 0 \\ 6 & 8 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 15 \\ 7 \\ 18 \end{bmatrix}$$

using the factorization of the coefficient matrix given in (7.2). In the permutation step we solve

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 15 \\ 7 \\ 18 \end{bmatrix},$$

which yields $v = (18, 7, 15)$. Next, we solve

$$\begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 0 & 15/19 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 7 \\ 15 \end{bmatrix}$$

by forward substitution. The solution is $w = (18, 1, 270/19)$. Finally, we solve

$$\begin{bmatrix} 6 & 8 & 8 \\ 0 & 19/3 & -8/3 \\ 0 & 0 & 135/19 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 1 \\ 270/19 \end{bmatrix}$$

by backward substitution, and find the solution $x = (-1, 1, 2)$.

Equations with multiple right-hand sides Multiple sets of linear equations with different right-hand sides,

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

with a general nonsingular coefficient matrix A, can be solved in

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

flops, 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 right-hand sides, at essentially the same cost as solving one.

7.4 Inverse of a nonsingular matrix

If A is nonsingular with LU factorization $A = PLU$ then its inverse can be expressed as

$$A^{-1} = U^{-1}L^{-1}P^T.$$

This expression gives another interpretation of algorithm 7.1. In steps 2–4, we evaluate $x = A^{-1}b = U^{-1}L^{-1}P^Tb$ by first computing $v = P^Tb$, then $w = L^{-1}v$, and finally $x = U^{-1}w$.

Computing the inverse The inverse A^{-1} can be computed by solving the matrix equation

$$AX = I$$

or, equivalently, the n equations $Ax_i = e_i$, where x_i is the ith column of A^{-1}, and e_i is the ith unit vector. This requires one LU factorization and n pairs of forward and backward substitutions. The cost is $(2/3)n^3 + n(2n^2) = (8/3)n^3$ flops, or about $3n^3$ flops.

7.5 Computing the LU factorization without pivoting

Before we describe the general algorithm for LU factorization, it is useful to consider the simpler factorization

$$A = LU$$

where L is unit lower triangular and U is upper triangular and nonsingular. We refer to this factorization as the *LU factorization without permutations* (or without pivoting). It is a special case of the general LU factorization with $P = I$.

To see how we can calculate L and U, we partition the matrices on both sides of $A = LU$ as

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix},$$

where a_{11} and u_{11} are scalars and the other dimensions follow from the conventions of block matrix notation. (For example, A_{12} is $1 \times (n-1)$, A_{21} is $(n-1) \times 1$, A_{22}

is $(n-1) \times (n-1)$, etc.). The matrix L_{22} is unit lower triangular, and U_{22} is upper triangular.

If we work out the product on the right, we obtain

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & U_{12} \\ u_{11}L_{21} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}.$$

Equating both sides allows us to determine the first row of U and the first column of L:

$$u_{11} = a_{11}, \qquad U_{12} = A_{12}, \qquad L_{21} = \frac{1}{a_{11}} A_{21}.$$

Furthermore,

$$L_{22}U_{22} = A_{22} - L_{21}U_{12} = A_{22} - \frac{1}{a_{11}} A_{21}A_{12},$$

so we can calculate L_{22} and U_{22} by factoring $A_{22} - L_{21}U_{12}$ as $A_{22} - L_{21}U_{12} = L_{22}U_{22}$, which is an LU factorization of dimension $n-1$. This suggests a recursive algorithm: to factor a matrix of order n, we calculate the first column of L and the first row of U, and then factor a matrix of order $n-1$. Continuing recursively, we arrive at a factorization of a 1×1 matrix, which is trivial.

Algorithm 7.2. LU FACTORIZATION WITHOUT PIVOTING.

given an $n \times n$-matrix A that can be factored as $A = LU$.

1. Calculate the first row of U: $u_{11} = a_{11}$ and $U_{12} = A_{12}$.
2. Calculate the first column of L: $L_{12} = (1/a_{11})A_{21}$.
3. Calculate the LU factorization $A_{22} - L_{21}U_{12} = L_{22}U_{22}$.

It can be shown that the cost is $(2/3)n^3$ flops.

Example As an example, let us factor the matrix

$$A = \begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix},$$

as $A = LU$, *i.e.*,

$$A = \begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

We start by calculating the first row of U and the first column of L:

$$\begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 3/4 & l_{32} & 1 \end{bmatrix} \begin{bmatrix} 8 & 2 & 9 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

The remaining elements are obtained from the identity

$$\begin{aligned}\begin{bmatrix} 9 & 4 \\ 7 & 9 \end{bmatrix} &= \begin{bmatrix} 1/2 & 1 & 0 \\ 3/4 & l_{32} & 1 \end{bmatrix} \begin{bmatrix} 2 & 9 \\ u_{22} & u_{23} \\ 0 & u_{33} \end{bmatrix} \\ &= \begin{bmatrix} 1/2 \\ 3/4 \end{bmatrix} \begin{bmatrix} 2 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{22} & u_{23} \\ 0 & u_{33} \end{bmatrix}\end{aligned}$$

or

$$\begin{bmatrix} 8 & -1/2 \\ 11/2 & 9/4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{22} & u_{23} \\ 0 & u_{33} \end{bmatrix}.$$

This is a 2×2 LU factorization. Again it is easy to find the first rows and columns of the triangular matrices on the right:

$$\begin{bmatrix} 8 & -1/2 \\ 11/2 & 9/4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 11/16 & 1 \end{bmatrix} \begin{bmatrix} 8 & -1/2 \\ 0 & u_{33} \end{bmatrix}.$$

Finally, the remaining element u_{33} follows from

$$9/4 = -(11/16) \cdot (1/2) + u_{33},$$

i.e., $u_{33} = 83/32$. Putting everything together, we obtain the following factorization:

$$A = \begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 3/4 & 11/16 & 1 \end{bmatrix} \begin{bmatrix} 8 & 2 & 9 \\ 0 & 8 & -1/2 \\ 0 & 0 & 83/32 \end{bmatrix}.$$

Existence of LU factorization without permutations Simple examples show that not every nonsingular matrix can be factored as $A = LU$. The matrix

$$A = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix},$$

for example, is nonsingular, but it cannot be factored as

$$\begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}.$$

If we apply algorithm 7.2, we first find $u_{11} = 0$, $u_{12} = 1$. Hence, l_{21} must satisfy $1 = l_{21} \cdot 0$, which is impossible. In this example the factorization algorithm breaks down in the first step because $a_{11} = 0$. Even if the first step succeeds ($a_{11} \neq 0$), we might run into the same problem (division by zero) in any of the subsequent recursive steps.

7.6 Computing the LU factorization (with pivoting)

We now extend algorithm 7.2 to the full LU factorization $A = PLU$. We will see that such a factorization exists if A is nonsingular, and we will describe a recursive algorithm to compute it.

We first note that a nonsingular 1×1-matrix A is simply a nonzero scalar, so its LU factorization is trivial: $A = PLU$ with $P = 1$, $L = 1$, $U = A$. Next we show that if it is true that every nonsingular $(n-1) \times (n-1)$ matrix has an LU factorization, then the same is true for nonsingular $n \times n$-matrices. By induction, this proves that any nonsingular matrix has an LU factorization.

Let us therefore assume that every nonsingular matrix of dimension $n-1$ has an LU factorization. Suppose we want to factor a nonsingular matrix A of order n. At least one element in the first column of A must be nonzero (since a matrix with a zero column is singular). We can therefore apply a row permutation that makes the 1,1 element nonzero. In other words, there exists a permutation matrix P_1 such that the matrix $\tilde{A} = P_1^T A$ satisfies $\tilde{a}_{11} \neq 0$. We partition the permuted matrix $\tilde{A}$ as

$$\tilde{A} = P_1^T A = \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix},$$

where $\tilde{A}_{22}$is has size $(n-1) \times (n-1)$ (and $\tilde{a}_{11} \neq 0$).

Now consider the matrix

$$\tilde{A}_{22} - \frac{1}{\tilde{a}_{11}} \tilde{A}_{21} \tilde{A}_{12}. \tag{7.3}$$

This is the Schur complement of $\tilde{a}_{11}$ in $\tilde{A}$ and we know (from section 3.3.1) that the Schur complement is nonsingular if $\tilde{A}$ is nonsingular. By assumption, this means it can be factored as

$$\tilde{A}_{22} - \frac{1}{\tilde{a}_{11}} \tilde{A}_{21} \tilde{A}_{12} = P_2 L_{22} U_{22}.$$

This provides the desired LU factorization of A:

$$\begin{aligned} A &= P_1 \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix} \\ &= P_1 \begin{bmatrix} 1 & 0 \\ 0 & P_2 \end{bmatrix} \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ P_2^T \tilde{A}_{21} & P_2^T \tilde{A}_{22} \end{bmatrix} \\ &= P_1 \begin{bmatrix} 1 & 0 \\ 0 & P_2 \end{bmatrix} \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ P_2^T \tilde{A}_{21} & L_{22} U_{22} + (1/\tilde{a}_{11}) P_2^T \tilde{A}_{21} \tilde{A}_{12} \end{bmatrix} \\ &= P_1 \begin{bmatrix} 1 & 0 \\ 0 & P_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ (1/\tilde{a}_{11}) P_2^T \tilde{A}_{21} & L_{22} \end{bmatrix} \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ 0 & U_{22} \end{bmatrix} \end{aligned}$$

so if we define

$$P = P_1 \begin{bmatrix} 1 & 0 \\ 0 & P_2 \end{bmatrix}, \qquad L = \begin{bmatrix} 1 & 0 \\ (1/\tilde{a}_{11}) P_2^T \tilde{A}_{21} & L_{22} \end{bmatrix}, \qquad U = \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ 0 & U_{22} \end{bmatrix}$$

then P is a permutation matrix, L is unit lower triangular, U is upper triangular, and $A = PLU$.

We summarize the idea by stating the algorithm recursively.

Algorithm 7.3. LU FACTORIZATION.

given a nonsingular $n \times n$-matrix A.

1. Choose P_1 such that $\tilde{A} = P_1^T A$ satisfies $\tilde{a}_{11} \neq 0$.
2. Compute the LU factorization $\tilde{A}_{22} - (1/\tilde{a}_{11})\tilde{A}_{21}\tilde{A}_{12} = P_2 L_{22} U_{22}$.
3. The LU factorization of A is $A = PLU$ with

$$P = P_1 \begin{bmatrix} 1 & 0 \\ 0 & P_2 \end{bmatrix}, \qquad L = \begin{bmatrix} 1 & 0 \\ (1/\tilde{a}_{11})P_2^T \tilde{A}_{21} & L_{22} \end{bmatrix}, \qquad U = \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ 0 & U_{22} \end{bmatrix}.$$

It can be shown that the total cost is $(2/3)n^3$ flops.

Example Let us factor the matrix

$$A = \begin{bmatrix} 0 & 5 & 5 \\ 2 & 3 & 0 \\ 6 & 9 & 8 \end{bmatrix}.$$

Since $a_{11} = 0$ we need a permutation that brings the second row or the third row in the first position. For example, we can choose

$$P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

and proceed with the factorization of

$$\tilde{A} = P_1^T A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 5 & 5 \\ 2 & 3 & 0 \\ 6 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 6 & 9 & 8 \\ 2 & 3 & 0 \\ 0 & 5 & 5 \end{bmatrix}.$$

To factor $\tilde{A}$ we need the LU factorization of

$$\begin{aligned} A^{(1)} = \tilde{A}_{22} - (1/\tilde{a}_{11})\tilde{A}_{21}\tilde{A}_{12} &= \begin{bmatrix} 3 & 0 \\ 5 & 5 \end{bmatrix} - (1/6) \begin{bmatrix} 2 \\ 0 \end{bmatrix} \begin{bmatrix} 9 & 8 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -8/3 \\ 5 & 5 \end{bmatrix}. \end{aligned}$$

The first element is zero, so we need to apply a permutation

$$P_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

in order to switch the two rows. We denote the permuted matrix as $\tilde{A}^{(1)}$:

$$\tilde{A}^{(1)} = P_2^T A^{(1)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -8/3 \\ 5 & 5 \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 0 & -8/3 \end{bmatrix}.$$

This matrix is upper triangular, so we do not have to go to the next level of the recursion to find its factorization: $\tilde{A}^{(1)} = L_{22}U_{22}$ where

$$L_{22} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad U_{22} = \begin{bmatrix} 5 & 5 \\ 0 & -8/3 \end{bmatrix}.$$

We can now assemble the LU factorization $A = PLU$, as in step 3 of the algorithm outline. The permutation matrix is

$$P = P_1 \begin{bmatrix} 1 & 0 \\ 0 & P_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

The lower triangular matrix is

$$L = \begin{bmatrix} 1 & 0 \\ (1/\tilde{a}_{11})P_2^T\tilde{A}_{21} & L_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1/3 & 0 & 1 \end{bmatrix}.$$

The upper triangular matrix is

$$U = \begin{bmatrix} \tilde{a}_{11} & \tilde{A}_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} 6 & 9 & 8 \\ 0 & 5 & 5 \\ 0 & 0 & -8/3 \end{bmatrix}.$$

In summary, the LU factorization of A is

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 3 & 0 \\ 6 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1/3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 & 9 & 8 \\ 0 & 5 & 5 \\ 0 & 0 & -8/3 \end{bmatrix}.$$

7.7 Effect of rounding error

In this section we discuss the effect of rounding errors on the accuracy of the LU factorization method for solving linear equations.

As an example, we consider two equations in two variables

$$\begin{aligned} 10^{-5}x_1 + x_2 &= 1 \\ x_1 + x_2 &= 0. \end{aligned}$$

The solution is

$$x_1 = \frac{-1}{1 - 10^{-5}}, \qquad x_2 = \frac{1}{1 - 10^{-5}} \tag{7.4}$$

as is easily verified by substitution. We will solve the equations using the LU factorization, and introduce small errors by rounding some of the intermediate results to four significant decimal digits.

The matrix

$$A = \begin{bmatrix} 10^{-5} & 1 \\ 1 & 1 \end{bmatrix}$$

has two LU factorizations:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 10^5 & 1 \end{bmatrix} \begin{bmatrix} 10^{-5} & 1 \\ 0 & 1 - 10^5 \end{bmatrix} \tag{7.5}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 10^{-5} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 - 10^{-5} \end{bmatrix}. \tag{7.6}$$

Suppose we use the first factorization, and round the $2, 2$ element of U,

$$u_{22} = 1 - 10^5 = -0.99999 \cdot 10^5,$$

to four significant digits, *i.e.*, replace it with $-1.0000 \cdot 10^5 = -10^5$. (The other elements of L and U do not change if we round them to four significant digits.) We proceed with the solve step, using the approximate factorization

$$A \approx \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 10^5 & 1 \end{bmatrix} \begin{bmatrix} 10^{-5} & 1 \\ 0 & -10^5 \end{bmatrix}.$$

In the forward substitution step we solve

$$\begin{bmatrix} 1 & 0 \\ 10^5 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

which gives $z_1 = 1$, $z_2 = -10^5$. In the backward substitution step we solve

$$\begin{bmatrix} 10^{-5} & 1 \\ 0 & -10^5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -10^5 \end{bmatrix}.$$

The solution is $x_1 = 0$, $x_2 = 1$. Comparing this with the exact solution (7.4), we see that the error in x_1 is 100%! One small rounding error (replacing -0.99999 by 1) has caused a huge error in the result.

This phenomenon is called *numerical instability*. An algorithm is numerically unstable if small rounding errors can cause a very large error in the result. The example shows that solving linear equations using the LU factorization can be numerically unstable.

In a similar way, we can examine the effect of rounding errors on the second factorization (7.6). Suppose we round the $2, 2$ element $u_{22} = 1 - 10^{-5} = 0.99999$ to 1, and use the approximate factorization

$$A \approx \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 10^{-5} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

In the forward substitution step we solve

$$\begin{bmatrix} 1 & 0 \\ 10^{-5} & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

which gives $z_1 = 0$, $z_2 = 1$. Solving

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

by backward substitution, yields $x_1 = -1$, $x_2 = 1$. In this case the error in the result is very small (about 10^{-5}, *i.e.*, of the same order as the rounding error that we introduced).

We conclude that choosing the wrong permutation matrix P can have a disastrous effect on the accuracy of the solution.

A careful error analysis and extensive practical experience have provided a good remedy for numerical instability in the LU factorization: when selecting the permutation matrix P in step 1 of the algorithm of section 7.6, choose P so that $\tilde{a}_{11}$ is the element with the largest absolute value in the first column of A. (This is consistent with our observation in the small example: the largest element of the first column of A is 1, so according to this guideline, we should permute the two rows of A.) With this selection rule for P, the algorithm is stable, except in rare cases.

7.8 Sparse linear equations

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

$$A = P_1 L U P_2, \tag{7.7}$$

where P_1 and P_2 are permutation matrices, L is unit 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 $Ax = b$.

Sparse LU factorization algorithms must take into account two criteria when selecting the permutation matrices P_1 and P_2:

1. *Numerical stability.* As we have seen in section 7.7 the LU factorization algorithm may be numerically unstable (or may not exist) for certain choices of the permutation matrix. The same is true for the factorization (7.7).

2. *Sparsity of L and U.* The sparsity of the factors L and U depends on the permutations P_1 and P_2, which have to be chosen in a way that tends to yield sparse L and U.

Achieving a good compromise between these two objectives is quite difficult, and codes for sparse LU factorization are much more complicated than codes for dense LU factorization.

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.

Exercises

7.1 (a) For what values of a_1, a_2, ..., a_n is the $n \times n$-matrix

$$A = \begin{bmatrix} a_1 & 1 & 0 & \cdots & 0 & 0 \\ a_2 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & 0 & 0 & \cdots & 1 & 0 \\ a_{n-1} & 0 & 0 & \cdots & 0 & 1 \\ a_n & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

nonsingular?

(b) Assuming A is nonsingular, how many floating-point operations (flops) do you need to solve $Ax = b$?

(c) Assuming A is nonsingular, what is the inverse A^{-1}?

7.2 Consider the set of linear equations

$$(D + uv^T)x = b$$

where u, v, and b are given n-vectors, and D is a given diagonal matrix. The diagonal elements of D are nonzero and $u^T D^{-1} v \neq -1$. (This implies that the matrix $D + uv^T$ is nonsingular.)

(a) What is the cost of solving these equations using the following method?

- First calculate $A = D + uv^T$.
- Then solve $Ax = b$ using the standard method (via LU factorization, which costs $(2/3)n^3$ flops).

(b) In exercise 3.11, we derive the following expression for the inverse of $D + uv^T$:

$$(D + uv^T)^{-1} = D^{-1} - \frac{1}{1 + v^T D^{-1} u} D^{-1} u v^T D^{-1}.$$

This means we can also solve the equations by evaluating $x = (D + uv^T)^{-1} b$, using the expression for the inverse. What is the cost of this method?

7.3 For each subproblem, we give a naive but correct algorithm, in MATLAB notation. Derive a flop count, assuming that matrix inverses and solutions of linear equations are computed using the factor-solve method of section 7.1. Use the values $f = (2/3)n^3$, $s = 2n^2$ for the flop counts of the factorization and solve steps (for a set of n linear equations in n variables). Assume the cost of computing the inverse of an $n \times n$ matrix is $f + ns = (8/3)n^3$ flops.

If possible, give a more efficient method. You do not have to provide any MATLAB code, as long as the description of your method is clear. If you know several methods, give the most efficient one.

(a) Calculate $c^T A^{-1} b$ where $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{n \times n}$, and $b \in \mathbf{R}^n$ are given. The matrix A is nonsingular.

```
val = c'*(inv(A)*b)
```

(b) Calculate $c^T A^{-1} B$ where $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{n \times n}$, and $B \in \mathbf{R}^{n \times m}$ are given. The matrix A is nonsingular.

```
val = c'*(inv(A)*B)
```

(c) Solve the set of equations

$$\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} x = \begin{bmatrix} b \\ c \end{bmatrix},$$

where $A \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{n\times n}$, $b \in \mathbf{R}^n$, and $c \in \mathbf{R}^n$ are given, and 0 is the zero matrix of dimension $n \times n$. The matrices A and B are nonsingular.

```
x = [A zeros(n,n); zeros(n,n) B ] \ [b; c]
```

(d) Solve the set of equations

$$\begin{bmatrix} A & B \\ C & I \end{bmatrix} x = \begin{bmatrix} b \\ c \end{bmatrix},$$

where $A \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{n\times 10n}$, $C \in \mathbf{R}^{10n\times n}$, $b \in \mathbf{R}^n$, and $c \in \mathbf{R}^{10n}$ are given, and I is the identity matrix of dimension $10n \times 10n$. The matrix

$$\begin{bmatrix} A & B \\ C & I \end{bmatrix}$$

is nonsingular.

```
x = [A, B; C, eye(10*n)] \ [b; c]
```

(e) Solve the set of equations

$$\begin{bmatrix} I & B \\ C & I \end{bmatrix} x = \begin{bmatrix} b \\ c \end{bmatrix},$$

where B is $m \times n$, C is $n \times m$, and $n > m$. The matrix

$$\begin{bmatrix} I & B \\ C & I \end{bmatrix} x = \begin{bmatrix} b \\ c \end{bmatrix}$$

is nonsingular.

```
x = [eye(m), B; C, eye(n)] \ [b; c]
```

7.4 A diagonal matrix with diagonal elements $+1$ or -1 is called a *signature matrix*. The matrix

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

is an example of a 3×3 signature matrix. If S is a signature matrix, and A is a square matrix that satisfies

$$A^T S A = S, \tag{7.8}$$

then we say that A is *pseudo-orthogonal* with respect to S.

(a) Suppose S is an $n \times n$ signature matrix, and u is an n-vector with $u^T S u \neq 0$. Show that the matrix

$$A = S - \frac{2}{u^T S u} u u^T$$

is pseudo-orthogonal with respect to S.

(b) Show that pseudo-orthogonal matrices are nonsingular. In other words, show that any square matrix A that satisfies (7.8) for some signature matrix S is nonsingular.

(c) Describe an efficient method for solving $Ax = b$ when A is pseudo-orthogonal. 'Efficient' here means that the complexity is at least an order of magnitude less than the $(2/3)n^3$ complexity of the standard method for a general set of linear equations. Give the complexity of your method (number of flops for large n).

(d) Show that if A satisfies (7.8) then $ASA^T = S$. In other words, if A is pseudo-orthogonal with respect to S, then A^T is also pseudo-orthogonal with respect to S.

7.5 Calculate the LU factorization without pivoting of the matrix

$$A = \begin{bmatrix} -3 & 2 & 0 & 3 \\ 6 & -6 & 0 & -12 \\ -3 & 6 & -1 & 16 \\ 12 & -14 & -2 & -25 \end{bmatrix}.$$

You can check your result in MATLAB, but you have to provide the details of your calculation.

7.6 You are given a nonsingular $n \times n$-matrix A and an n-vector b. You are asked to evaluate

$$x = (I + A^{-1} + A^{-2} + A^{-3})\, b$$

where $A^{-2} = (A^2)^{-1}$ and $A^{-3} = (A^3)^{-1}$.

Describe in detail how you would compute x, and give the flop counts of the different steps in your algorithm. If you know several methods, give the most efficient one (least number of flops for large n).

7.7 Suppose you are asked to solve K sets of linear equations

$$\begin{aligned} AD_1Bx_1 &= b_1 \\ AD_2Bx_2 &= b_2 \\ &\vdots \\ AD_KBx_K &= b_K. \end{aligned}$$

The $n \times n$-matrices A and B are nonsingular and given. The matrices D_k are diagonal with nonzero diagonal elements. The right-hand sides $b_k \in \mathbf{R}^n$ are also given. The variables in the problem are the K n-vectors x_k, $k = 1, \ldots, K$.

Describe an efficient method for computing the vectors x_k. Compare with the cost of solving K sets of linear equations of size $n \times n$, using a standard method.

7.8 What is the most efficient way to compute the following $n \times n$-matrices X? Justify your answer by giving the number of flops, assuming n is large. The vectors u and v have size n, and the matrix A is $n \times n$. A is nonsingular.

(a) $X = vu^T Avu^T$.

(b) $X = vu^T A^{-1} vu^T$.

(c) $X = vu^T (A + A^{-1}) vu^T$.

7.9 Suppose you have to solve two sets of linear equations

$$Ax_1 = b_1, \qquad A^T x_2 = b_2$$

where A is $n \times n$, b_1 and b_2 are n-vectors, and A is nonsingular. The unknowns are the n-vectors x_1 and x_2.

What is the most efficient way to solve these two problems (for general A, *i.e.*, when A has no special structure that you can take advantage of)? You do not have to give any code, but you must say clearly what the different steps in your algorithm are, and how many flops they cost.

7.10 Suppose you have to solve two sets of linear equations

$$Ax = b, \qquad (A + uv^T)y = b,$$

where A is $n \times n$ and given, and u, v, and b are given n-vectors. The variables are x and y. We assume that A and $A + uv^T$ are nonsingular.
The cost of solving the two systems from scratch is $(4/3)n^3$ flops. Give a more efficient method, based on the expression

$$(A + uv^T)^{-1} = A^{-1} - \frac{1}{1 + v^T A^{-1} u} A^{-1} u v^T A^{-1}$$

(see exercise 3.11). Clearly state the different steps in your algorithm and give the flop count of each step (for large n). What is the total flop count (for large n)?

7.11 Consider the equation

$$AXA^T = B$$

where A and B are given $n \times n$-matrices, with A nonsingular. The variables are the n^2 elements of the $n \times n$-matrix X.

(a) Prove that there is a unique solution X.

(b) Can you give an efficient algorithm for computing X, based on factoring A and/or A^T? What is the cost of your algorithm (number of flops for large n)?

7.12 Assume A is a nonsingular matrix of order n. In exercise 3.12, we have seen that the inverse of the matrix

$$M = \begin{bmatrix} A & A + A^{-T} \\ A & A \end{bmatrix} \tag{7.9}$$

is given by

$$M^{-1} = \begin{bmatrix} -A^T & A^{-1} + A^T \\ A^T & -A^T \end{bmatrix}. \tag{7.10}$$

(a) Compare the cost (number of flops for large n) of the following two methods for solving a set of linear equations $Mx = b$, given A and b.

Method 1. Calculate A^{-1}, build the matrix M as defined in equation (7.9), and solve $Mx = b$ using the standard method This method would correspond to the MATLAB code `x = [A  A+inv(A)'; A A] \ b`.

Method 2. Calculate A^{-1}, build the matrix M^{-1} as defined in equation (7.10), and form the matrix vector product $x = M^{-1}b$. This method would correspond to the MATLAB code`x = [-A' inv(A)+A'; A'  -A']*b`.

(b) Can you improve the fastest of the two methods described in part (a)? (You can state your answer in the form of an improved version of the MATLAB code given in part (a), but that is not necessary, as long as the steps in your method are clear.)

7.13 Consider the set of linear equations

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{7.11}$$

where the $m \times m$-matrices A_{ij} and the vectors m-vectors b_1, b_2, b_3 are given. The variables are the three m-vectors x_1, x_2, x_3. In other words we have $n = 3m$ equations in $3m$ variables. We assume that the matrices A_{11}, A_{22}, A_{33} are nonsingular.

(a) Describe an efficient method for solving (7.11). You do not have to write any MATLAB code, but you should state clearly what the different steps in your method are, and how many flops they cost. If you know several methods, you should select the most efficient one.

(b) Same question, assuming that $A_{11} = A_{22} = A_{33}$.

(c) Can you extend the algorithm of parts (a) and (b) to equations of the form

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1,K-2} & A_{1,K-1} & A_{1K} \\ 0 & A_{22} & \cdots & A_{2,K-2} & A_{2,K-1} & A_{2K} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & A_{K-2,K-2} & A_{K-2,K-1} & A_{K-2,K} \\ 0 & 0 & \cdots & 0 & A_{K-1,K-1} & A_{K-1,K} \\ 0 & 0 & \cdots & 0 & 0 & A_{KK} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{K-2} \\ x_{K-1} \\ x_K \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{K-2} \\ b_{K-1} \\ b_K \end{bmatrix}$$

where A_{ij} is $m \times m$ and b_i is an m-vector. The variables are the m-vectors x_i. The diagonal blocks A_{ii} are nonsingular. Compare the cost of your algorithm with the cost of a standard method for solving Km equations in Km variables.

7.14 Describe an efficient method for solving the equation

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ I & 0 & D \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b \\ c \\ d \end{bmatrix}.$$

The nine blocks in the coefficient matrix have size $n \times n$. The matrix A is nonsingular, and the matrix D is diagonal with nonzero diagonal elements. The vectors b, c, and d in the right-hand side are n-vectors. The variables are the n-vectors x, y, z.

If you know several methods, give the most efficient one. Clearly state the different steps in your algorithm, give the complexity (number of flops) of each step, and the total number of flops.

7.15 We define a $2m \times 2m$ matrix

$$B = \begin{bmatrix} -2A & 4A \\ 3A & -5A \end{bmatrix},$$

where A is a nonsingular $m \times m$ matrix.

(a) Express the inverse of B in terms of A^{-1}.

(b) The cost of solving the linear equations

$$\begin{bmatrix} -2A & 4A \\ 3A & -5A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

with variables $x_1 \in \mathbf{R}^m$, $x_2 \in \mathbf{R}^m$, using the standard method (*i.e.*, using the command

```
[-2*A  4*A; 3*A  -5*A] \ [b1; b2]
```

in MATLAB), is $(2/3)(2m)^3 = (16/3)m^3$ flops for large m.

Formulate a more efficient method. Clearly state the different steps in your algorithm and give the cost (number of flops for large m) of each step, as well as the total flop count. If you know several methods, give the most efficient one.

7.16 For each subproblem, describe an efficient method to evaluate the expression, and give a flop count. A is a nonsingular $n \times n$ matrix, and v_i, w_i, $i = 1, \ldots, m$, are n-vectors.

(a) $\sum_{i=1}^{m} v_i^T A^{-1} w_i$

(b) $\sum_{i=1}^{m} v_i^T (A + A^{-1}) w_i$

(c) $\sum_{i=1}^{m}\sum_{j=1}^{m} v_i^T A^{-1} w_j$

(d) $\sum_{i=1}^{m}\sum_{j=1}^{m} v_i w_j^T A^{-1} w_i v_j^T$

If you know several methods, choose the most efficient one. Include only the dominant terms in the flop counts, assuming m and n are large.

7.17 Let A be a nonsingular $n \times n$ matrix and b an n-vector. In each subproblem, describe an efficient method for computing the vector x and give a flop count of your algorithm, including terms of order two (n^2) and higher. If you know several methods, give the most efficient one (least number of flops for large n).

(a) $x = (A^{-1} + A^{-2})b$.

(b) $x = (A^{-1} + A^{-T})b$.

(c) $x = (A^{-1} + JA^{-1}J)b$ where J is the $n \times n$ matrix

$$J = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

(J is the identity matrix with its columns reversed: $J_{ij} = 1$ if $i + j = n + 1$ and $J_{ij} = 0$ otherwise.)

7.18 Suppose A and B are $n \times n$ matrices with A nonsingular, and b, c and d are vectors of length n. Describe an efficient algorithm for solving the set of linear equations

$$\begin{bmatrix} A & B & 0 \\ 0 & A^T & B \\ 0 & 0 & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$$

with variables $x_1 \in \mathbf{R}^n$, $x_2 \in \mathbf{R}^n$, $x_3 \in \mathbf{R}^n$. Give a flop count for your algorithm, including all terms that are cubic or quadratic in n. If you know several methods, give the most efficient one (least number of flops for large n).

7.19 Suppose A is positive definite matrix of order n.

(a) Show that the matrix

$$\begin{bmatrix} A & d \\ d^T & -1 \end{bmatrix}$$

is nonsingular, for all values of $d \in \mathbf{R}^n$.

(b) Formulate an efficient method for solving the two sets of equations

$$Ax = b, \qquad \begin{bmatrix} A & d \\ d^T & -1 \end{bmatrix} \begin{bmatrix} y \\ t \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

The variable in the first equation is an n-vector x; the variables in the second equation are the n-vector y and a scalar t. Your algorithm should return x, y and t, given A, b, and d.

Describe in detail the different steps in your method, and give a flop count of each step (for large n). If you know different methods, choose the most efficient one.

7.20 Consider the linear equation

$$(A + \epsilon B)x = b,$$

where A and B are given $n \times n$-matrices, b is a given n-vector, and ϵ is a scalar parameter. We assume that A is nonsingular, and therefore $A + \epsilon B$ is nonsingular for sufficiently small ϵ. The solution of the equation is

$$x(\epsilon) = (A + \epsilon B)^{-1}b,$$

a complicated nonlinear function of ϵ. In order to find a simple approximation of $x(\epsilon)$, valid for small ϵ, we can expand $x(\epsilon) = (A + \epsilon B)^{-1}b$ in a series

$$x(\epsilon) = x_0 + \epsilon x_1 + \epsilon^2 x_2 + \epsilon^3 x_3 + \cdots,$$

where x_0, x_1, x_2, x_3, ... are n-vectors, and then truncate the series after a few terms. To determine the coefficients x_i in the series, we examine the equation

$$(A + \epsilon B)(x_0 + \epsilon x_1 + \epsilon^2 x_2 + \epsilon^3 x_3 + \cdots) = b.$$

Expanding the product on the left-hand side gives

$$Ax_0 + \epsilon(Ax_1 + Bx_0) + \epsilon^2(Ax_2 + Bx_1) + \epsilon^3(Ax_3 + Bx_2) + \cdots = b.$$

We see that if this holds for all ϵ in a neighborhood of zero, the coefficients x_i must satisfy

$$Ax_0 = b, \qquad Ax_1 + Bx_0 = 0, \qquad Ax_2 + Bx_1 = 0, \qquad Ax_3 + Bx_2 = 0, \qquad \ldots. \quad (7.12)$$

Describe an efficient method for computing the first $k+1$ coefficients $x_0, \ldots, x_k$ from (7.12). What is the complexity of your method (number of flops for large n, assuming $k \ll n$)? If you know several methods, give the most efficient one.

7.21 Suppose A is a nonsingular $n \times n$-matrix.

(a) Show that the matrix

$$\begin{bmatrix} A & b \\ a^T & 1 \end{bmatrix}$$

is nonsingular if a and b are n-vectors that satisfy $a^T A^{-1} b \neq 1$.

(b) Suppose a_1, a_2, b_1, b_2 are n-vectors with $a_1^T A^{-1} b_1 \neq 1$ and $a_2^T A^{-1} b_2 \neq 1$. From part 1, this means that the coefficient matrices in the two equations

$$\begin{bmatrix} A & b_1 \\ a_1^T & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} c_1 \\ d_1 \end{bmatrix}, \qquad \begin{bmatrix} A & b_2 \\ a_2^T & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_2 \\ d_2 \end{bmatrix}$$

are nonsingular. Describe an efficient method for solving the two equations. The variables are the two n-vectors x_1 and x_2, and the two scalars y_1 and y_2.

If you know several methods, give the most efficient one. Take advantage of the fact that the 1,1 blocks of the two coefficient matrices are the same. What is the complexity of your method (number of flops for large n)?

7.22 Let A be a nonsingular $n \times n$-matrix and let u, v be two n-vectors that satisfy $v^T A^{-1} u \neq 1$.

(a) Show that

$$\begin{bmatrix} A & u \\ v^T & 1 \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{1 - v^T A^{-1} u} \begin{bmatrix} A^{-1}u \\ -1 \end{bmatrix} \begin{bmatrix} v^T A^{-1} & -1 \end{bmatrix}.$$

(b) Describe an efficient method for solving the two equations

$$Ax = b, \qquad \begin{bmatrix} A & u \\ v^T & 1 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}.$$

The variables are the n-vectors x and y, and the scalar z.

Describe in detail the different steps in your algorihm and give a flop count of each step. If you know several methods, choose the most efficient one (least number of flops for large n).

7.23 Explain how you can solve the following problems using a single LU factorization and without computing matrix inverses. The matrix A is a given nonsingular $n \times n$ matrix. For each problem, carefully explain the different steps in your algorithm, give the cost of each step, and the total cost (number of flops for large n, excluding the LU factorization itself). If you know several methods, give the most efficient one.

(a) Compute $\left(A^{-1} + A^{-T}\right)^2 b$, where b is a given n-vector.

(b) Solve the equation $AXA = B$ for the unknown X, where B is a given $n \times n$-matrix.

(c) Compute $\sum_{i=1}^{n} \sum_{j=1}^{n} (A^{-1})_{ij}$, the sum of all the entries of A^{-1}.

7.24 In this problem we examine the effect of reordering the rows and columns of a matrix A on the sparsity of the factors L and U in $A = LU$.

Download the MATLAB files `ch7ex24.m` and `mylu.m`. The command `A = ch7ex24` will create a 100×100-matrix of the form

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

where D is a diagonal matrix of dimension 99×99, and u and v are two vectors in $\mathbf{R}^{99}$. A is a sparse matrix with an 'arrow' structure. In MATLAB or Octave you can verify this by typing `spy(A)`, which displays a figure with the zero-nonzero pattern of A.

(a) Since MATLAB does not provide a function for the LU factorization without pivoting (the command `lu(A)` computes the factorization $A = PLU$), we will use `mylu.m`. The command `[L,U] = mylu(A)` calculates the factorization $A = LU$, if it exists.

Execute `[L,U] = mylu(A)`. What are the sparsity patterns of L and U?

(b) We can also factorize the matrix

$$B = \begin{bmatrix} D & v \\ u^T & 1 \end{bmatrix},$$

which can be obtained from A by reordering its rows and columns. In MATLAB, we can construct B as `B = A([2:100,1],[2:100,1])`. Factor B as $B = LU$. What are the sparsity patterns of L and U?

Chapter 8

Linear least-squares

In this and the next chapter we discuss overdetermined sets of linear equations

$$Ax = b,$$

where A is an $m \times n$-matrix, b is an m-vector, and $m > n$. An overdetermined set of linear equations usually does not have a solution, so we have to accept an approximate solution, and define what we mean by 'solving' the equations. The most widely used definition is the least-squares solution, which minimizes $\|Ax-b\|^2$.

In the first half of the chapter (sections 8.1–8.3) we define the least-squares problem and give some examples and applications. In the second half we characterize the solution and present an algorithm for its computation.

8.1 Definition

A vector $\hat{x}$ *minimizes* $\|Ax - b\|^2$ if $\|A\hat{x} - b\|^2 \leq \|Ax - b\|^2$ for all x. We use the notation

$$\text{minimize} \quad \|Ax - b\|^2 \tag{8.1}$$

to denote the problem of finding an $\hat{x}$ that minimizes $\|Ax - b\|^2$. This is called a *least-squares problem* (LS problem), for the following reason. Let $r_i(x)$ be the ith component of $Ax - b$:

$$r_i(x) = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - b_i, \quad i = 1, \ldots, m. \tag{8.2}$$

By definition of the Euclidean norm,

$$\|Ax - b\|^2 = \sum_{i=1}^{m} r_i(x)^2,$$

so the function that we minimize in (8.1) is the sum of the squares of m functions $r_i(x)$, hence the term least-squares. The problem is sometimes called *linear* least-squares to emphasize that each of the functions r_i in (8.2) is affine (meaning, a

linear function $\sum_j a_{ij}x_j$ plus a constant $-b_i$), and to distinguish it from *nonlinear* least-squares problems, in which we allow arbitrary functions r_i.

We sometimes omit the square in (8.1) and use the notation

$$\text{minimize} \quad \|Ax - b\|$$

instead. This is justified because if $\hat{x}$ minimizes $\|Ax - b\|^2$, then it also minimizes $\|Ax - b\|$.

Example We consider the least-squares problem defined by

$$A = \begin{bmatrix} 2 & 0 \\ -1 & 1 \\ 0 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

The set of three equations in two variables $Ax = b$,

$$2x_1 = 1, \qquad -x_1 + x_2 = 0, \qquad 2x_2 = -1,$$

has no solution. The corresponding least-squares problem is

$$\text{minimize} \quad (2x_1 - 1)^2 + (-x_1 + x_2)^2 + (2x_2 + 1)^2.$$

We can find the solution by setting the derivatives of the function

$$f(x) = (2x_1 - 1)^2 + (-x_1 + x_2)^2 + (2x_2 + 1)^2$$

equal to zero:

$$\begin{aligned} \frac{\partial f(x)}{\partial x_1} = 4(2x_1 - 1) - 2(-x_1 + x_2) &= 0 \\ \frac{\partial f(x)}{\partial x_2} = 2(-x_1 + x_2) + 4(2x_2 + 1) &= 0. \end{aligned}$$

This gives two equations in two variables

$$10x_1 - 2x_2 = 4, \qquad -2x_1 + 10x_2 = -4$$

with a unique solution $\hat{x} = (1/3, -1/3)$.

8.2 Data fitting

Example Figure 8.1 shows 40 points (t_i, y_i) in a plane (shown as circles). The solid line shows a function $g(t) = \alpha + \beta t$ that satisfies

$$g(t_i) = \alpha + \beta t_i \approx y_i, \quad i = 1, \ldots, 40.$$

To compute coefficients α and β that give a good fit, we first need to decide on an

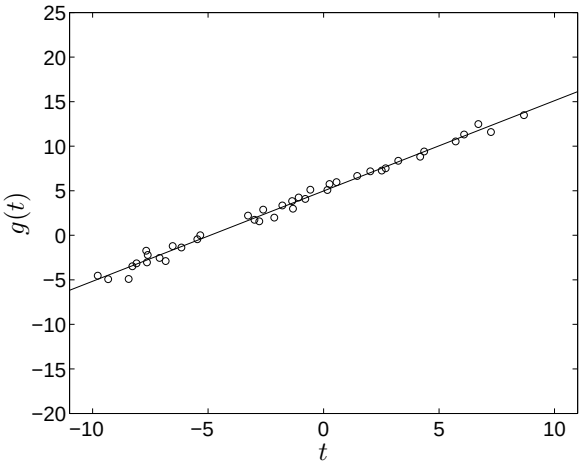


Figure 8.1 Least-squares fit of a straight line to 40 points in a plane.

approximation criterion or error function. In this example we use the error function

$$\sum_{i=1}^{40}(g(t_i)-y_i)^2 = \sum_{i=1}^{40}(\alpha+\beta t_i-y_i)^2.$$

This is the sum of squares of the errors $|g(t_i)-y_i|$. We then choose values of α and β that minimize this error function. This can be achieved by solving a least-squares problem: if we define

$$x = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_{40} \end{bmatrix}, \qquad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{40} \end{bmatrix},$$

then $\|Ax-b\|^2 = \sum_{i=1}^{40}(\alpha+\beta t_i-y_i)^2$, so we can minimize the error function by solving the least-squares problem

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

The MATLAB code for figure 8.1 is given below. (We assume that t and y are given as two 40×1 arrays `t` and `y`.)

```
x = [ones(40,1), t] \ y;
plot(t,y,'o', [-11; 11], [x(1)-11*x(2); x(1)+11*x(2)], '-');
```

Note that we use the command `x=A\b` to solve the least-squares problem (8.1).

Least-squares data fitting In a general data fitting problem, we are given m data points (t_i, y_i) where $y_i \in \mathbf{R}$, and we are asked to find a function $g(t)$ such that

$$g(t_i) \approx y_i, \quad i = 1, \ldots, m. \tag{8.3}$$

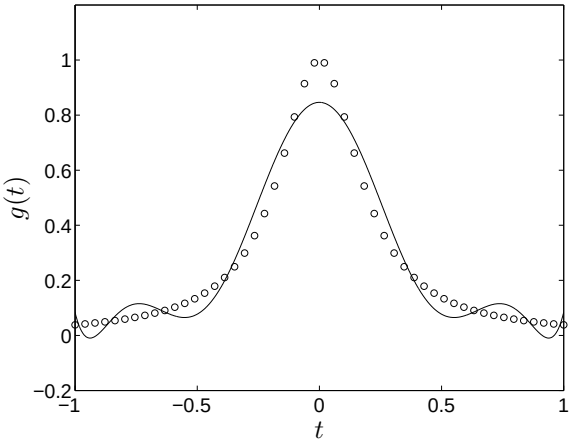


Figure 8.2 Least-squares polynomial fit of degree 9 (solid line) to 50 points (shown as circles).

Suppose we restrict g to functions of the form

$$g(t) = x_1 g_1(t) + x_2 g_2(t) + \cdots + x_n g_n(t),$$

where the functions $g_i(t)$ (called *basis functions*) are given, and the coefficients x_i are parameters to be determined. The simple fitting problem of figure 8.1 is an example with two basis functions $g_1(t) = 1$, $g_2(t) = t$.

We use

$$\sum_{i=1}^{m} (g(t_i) - y_i)^2 = \sum_{i=1}^{m} \left(x_1 g_1(t_i) + x_2 g_2(t_i) + \cdots + x_n g_n(t_i) - y_i\right)^2$$

to judge the quality of the fit (*i.e.*, the error in (8.3)), and determine the values of the coefficients x_i by minimizing the error function. This can be expressed as a least-squares problem with

$$A = \begin{bmatrix} g_1(t_1) & g_2(t_1) & \cdots & g_n(t_1) \\ g_1(t_2) & g_2(t_2) & \cdots & g_n(t_2) \\ \vdots & \vdots & & \vdots \\ g_1(t_m) & g_2(t_m) & \cdots & g_n(t_m) \end{bmatrix}, \qquad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Figure 8.2 shows an example in which we fit a polynomial

$$g(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_{10} t^9$$

to 50 points. (Here the basis functions are $g_1(t) = 1$, $g_2(t) = t$, ..., $g_{10}(t) = t^9$).

The MATLAB code is given below. We assume that the data points are given by two arrays `t` and `y` of length 50.

```
n = 10;
A = fliplr(vander(t));    % mxm matrix with elements A_(ij) = t_i^(j-1)
A = A(:,1:n);             % first n columns of A
x = A\y;

% to plot g, generate 1000 points ui between -1 and 1
u = linspace(-1,1,1000);
% evaluate g(ui) for all ui
g = x(n)*ones(1,1000);
for i = (n-1):-1:1
   g = g.*u + x(i);
end;
% plot data points and g
plot(u,g,'-', t, y, 'o');
```

8.3 Estimation

Suppose y is an m-vector of measurements, x is an n-vector of parameters to be estimated, and y and x are related as

$$y = Ax + w.$$

The $m \times n$-matrix A, which describes how the measured values depend on the unknown parameters, is given. The m-vector w is measurement error, and is unknown but presumably small. The estimation problem is to make a sensible guess as to what x is, given y.

If we guess that x has the value $\hat{x}$, then we are implicitly making the guess that w has the value $y - A\hat{x}$. If we assume that smaller values of w (measured by $\|w\|$) are more plausible than larger values, then a sensible choice for $\hat{x}$ is the least-squares solution, which minimizes $\|A\hat{x} - y\|$.

Example We are asked to estimate the position (u, v) of a point in a plane, given noisy measurements of the distances

$$\sqrt{(u - p_i)^2 + (v - q_i)^2}$$

of the point to four points with known positions (p_i, q_i). We denote the measured distances as ρ_i, $i = 1, 2, 3, 4$.

We assume hat $(u, v) \approx (0, 0)$, and use the approximation

$$\sqrt{(u - p_i)^2 + (v - q_i)^2} \approx \sqrt{p_i^2 + q_i^2} - \frac{up_i}{\sqrt{p_i^2 + q_i^2}} - \frac{vq_i}{\sqrt{p_i^2 + q_i^2}}, \tag{8.4}$$

which holds for small (u, v). With this approximation, we can write the measured distances as

$$\rho_i = \sqrt{p_i^2 + q_i^2} - \frac{p_i u}{\sqrt{p_i^2 + q_i^2}} - \frac{q_i v}{\sqrt{p_i^2 + q_i^2}} + w_i,$$

where the error w_i includes measurement error, as well as the error due to the linearization (8.4).

To compute estimates $\hat{u}$ and $\hat{v}$, given the measurements ρ_i and the positions (p_i, q_i), we can solve a least-squares problem with $x = (\hat{u}, \hat{v})$,

$$A = -\begin{bmatrix} p_1/\sqrt{p_1^2+q_1^2} & q_1/\sqrt{p_1^2+q_1^2} \\ p_2/\sqrt{p_2^2+q_2^2} & q_2/\sqrt{p_2^2+q_2^2} \\ p_3/\sqrt{p_3^2+q_3^2} & q_3/\sqrt{p_3^2+q_3^2} \\ p_4/\sqrt{p_4^2+q_4^2} & q_4/\sqrt{p_4^2+q_4^2} \end{bmatrix}, \qquad b = \begin{bmatrix} \rho_1 - \sqrt{p_1^2+q_1^2} \\ \rho_2 - \sqrt{p_2^2+q_2^2} \\ \rho_3 - \sqrt{p_3^2+q_3^2} \\ \rho_4 - \sqrt{p_4^2+q_4^2} \end{bmatrix}.$$

The MATLAB code below solves an example problem with

$$(p_1, q_1) = (10, 0), \quad (p_2, q_2) = (-10, 2), \quad (p_3, q_3) = (3, 9), \quad (p_4, q_4) = (10, 10),$$

and $(\rho_1, \rho_2, \rho_3, \rho_4) = (8.22, 11.9, 7.08, 11.33)$.

```
p = [10; -10; 3; 10];
q = [0; 2; 9; 10];
rho = [8.22; 11.9; 7.08; 11.33];

b = rho - sqrt(p.^2+q.^2);
A = -[p./sqrt(p.^2+q.^2)  q./sqrt(p.^2+q.^2)];
x = A\b;
```

The least-squares estimate is $x = (\hat{u}, \hat{v}) = (1.9676, 1.9064)$.

8.4 Solution of a least-squares problem

In this section we prove the following result. If A is left-invertible, then the solution of the least-squares problem

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

is unique and given by

$$\hat{x} = (A^T A)^{-1} A^T b. \tag{8.5}$$

(Recall from section 3.5 that $A^T A$ is positive definite if A is left-invertible.)

Proof We show that $\hat{x}$ satisfies

$$\|A\hat{x} - b\|^2 < \|Ax - b\|^2$$

for all $x \neq \hat{x}$. We start by writing

$$\begin{aligned} \|Ax - b\|^2 &= \|(Ax - A\hat{x}) + (A\hat{x} - b)\|^2 \\ &= \|Ax - A\hat{x}\|^2 + \|A\hat{x} - b\|^2 + 2(Ax - A\hat{x})^T(A\hat{x} - b), \end{aligned} \tag{8.6}$$

where we use

$$\|u+v\|^2 = (u+v)^T(u+v) = u^Tu + 2u^Tv + v^Tv = \|u\|^2 + \|v\|^2 + 2u^Tv.$$

The third term in (8.6) is zero:

$$\begin{aligned}(Ax - A\hat{x})^T(A\hat{x} - b) &= (x-\hat{x})^TA^T(A\hat{x} - b)\\ &= (x-\hat{x})^T(A^TA\hat{x} - A^Tb)\\ &= 0\end{aligned}$$

because $\hat{x} = (A^TA)^{-1}A^Tb$. With this simplification, (8.6) reduces to

$$\|Ax - b\|^2 = \|A(x-\hat{x})\|^2 + \|A\hat{x} - b\|^2.$$

The first term is nonnegative for all x, and therefore

$$\|Ax - b\|^2 \geq \|A\hat{x} - b\|^2.$$

Moreover we have equality only if $A(x-\hat{x}) = 0$, *i.e.*, $x = \hat{x}$ (because A has a zero nullspace). We conclude that $\|Ax-b\|^2 > \|A\hat{x}-b\|^2$ for all $x \neq \hat{x}$.

8.5 Solving least-squares problems by Cholesky factorization

The result in section 8.4 implies that if A is left-invertible, then we can compute the least-squares solution by solving a set of linear equations

$$(A^TA)x = A^Tb. \tag{8.7}$$

These equations are called the *normal equations* associated with the least-squares problem. Since A^TA is positive definite, we can use the Cholesky factorization for this purpose.

Algorithm 8.1. Solving least-squares problems by Cholesky factorization.

given a left-invertible $m \times n$-matrix A and an m-vector b.

1. Form $C = A^TA$ and $d = A^Tb$.
2. Compute the Cholesky factorization $C = LL^T$.
3. Solve $Lz = d$ by forward substitution.
4. Solve $L^Tx = z$ by backward substitution.

The cost of forming $C = A^TA$ in step 1 is mn^2 flops. (Note that C is symmetric, so we only need to compute the $(1/2)n(n+1) \approx (1/2)n^2$ elements in its lower triangular part.) The cost of calculating $d = A^Tb$ is $2mn$. The cost of step 2 is $(1/3)n^3$. Steps 3 and 4 cost n^2 each. The total cost of the algorithm is therefore $mn^2 + (1/3)n^3 + 2mn + 2n^2$ or roughly

$$mn^2 + (1/3)n^3.$$

Note that left-invertibility of A implies that $m \geq n$, so the most expensive step is the matrix-matrix multiplication A^TA, and not the Cholesky factorization.

Example We solve the least-squares problem with $m = 4$, $n = 3$, and

$$A = (1/5)\begin{bmatrix} 3 & -6 & 26 \\ 4 & -8 & -7 \\ 0 & 4 & 4 \\ 0 & -3 & -3 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \tag{8.8}$$

We first calculate $C = A^T A$ and $d = A^T b$:

$$C = \begin{bmatrix} 1 & -2 & 2 \\ -2 & 5 & -3 \\ 2 & -3 & 30 \end{bmatrix}, \qquad d = \begin{bmatrix} 7/5 \\ -13/5 \\ 4 \end{bmatrix}.$$

To solve the normal equations

$$\begin{bmatrix} 1 & -2 & 2 \\ -2 & 5 & -3 \\ 2 & -3 & 30 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7/5 \\ -13/5 \\ 4 \end{bmatrix},$$

we first compute the Cholesky factorization of C:

$$C = \begin{bmatrix} 1 & -2 & 2 \\ -2 & 5 & -3 \\ -2 & -3 & 30 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 2 & 1 & 5 \end{bmatrix}\begin{bmatrix} 1 & -2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 5 \end{bmatrix}.$$

We then solve

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 2 & 1 & 5 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 7/5 \\ -13/5 \\ 4 \end{bmatrix},$$

by forward substitution and find $(z_1, z_2, z_3) = (7/5, 1/5, 1/5)$. Finally we solve

$$\begin{bmatrix} 1 & -2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7/5 \\ 1/5 \\ 1/5 \end{bmatrix}$$

by back substitution. The solution is $x = (41/25, 4/25, 1/25)$.

Exercises

Definition

8.1 Formulate the following problems as least-squares problems. For each problem, give a matrix A and a vector b, such that the problem can be expressed as

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

(You do not have to solve the problems.)

(a) Minimize $x_1^2 + 2x_2^2 + 3x_3^2 + (x_1 - x_2 + x_3 - 1)^2 + (-x_1 - 4x_2 + 2)^2$.

(b) Minimize $(-6x_2 + 4)^2 + (-4x_1 + 3x_2 - 1)^2 + (x_1 + 8x_2 - 3)^2$.

(c) Minimize $2(-6x_2 + 4)^2 + 3(-4x_1 + 3x_2 - 1)^2 + 4(x_1 + 8x_2 - 3)^2$.

(d) Minimize $x^T x + \|Bx - d\|^2$ where the $p \times n$-matrix B and the p-vector d are given.

(e) Minimize $\|Bx - d\|^2 + 2\|Fx - g\|^2$. The $p \times n$-matrix B, the $l \times n$-matrix F, the p-vector d and the l-vector g are given.

(f) Minimize $x^T Dx + \|Bx - d\|^2$. D is a $n \times n$ diagonal matrix with positive diagonal elements, B is $p \times n$, and d is a p-vector. D, B and D are given.

8.2 Formulate the following problem as a least-squares problem. Find a polynomial

$$p(t) = x_1 + x_2 t + x_3 t^2 + x_4 t^3$$

that satisfies the following conditions.

- The values $p(t_i)$ at 4 given points t_i in the interval $[0, 1]$ should be approximately equal to given values y_i:
$$p(t_i) \approx y_i, \quad i = 1, \ldots, 4.$$
The points t_i are given and distinct ($t_i \neq t_j$ for $i \neq j$). The values y_i are also given.
- The derivatives of p at $t = 0$ and $t = 1$ should be small:
$$p'(0) \approx 0, \qquad p'(1) \approx 0.$$
- The average value of p over the interval $[0, 1]$ should be approximately equal to the value at $t = 1/2$:
$$\int_0^1 p(t)\, dt \approx p(1/2).$$

To determine coefficients x_i that satisfy these conditions, we minimize

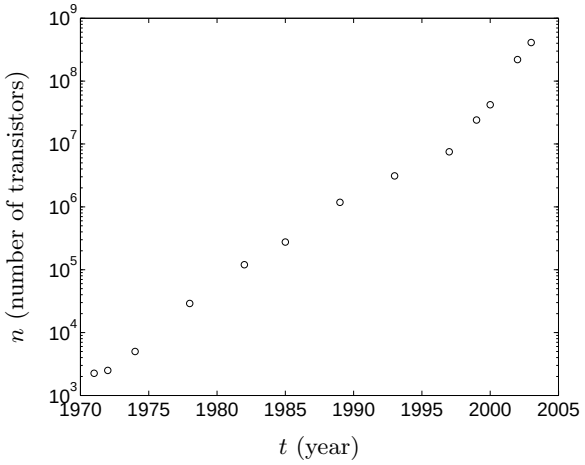
$$E(x) = \frac{1}{4}\sum_{i=1}^{4}(p(t_i) - y_i)^2 + p'(0)^2 + p'(1)^2 + \left(\int_0^1 p(t)\, dt - p(1/2)\right)^2.$$

Give A and b such that $E(x) = \|Ax - b\|^2$. Clearly state the dimensions of A and b, and what their elements are.

Applications

8.3 *Moore's law.* The figure and the table show the number of transistors in 13 microprocessors, and the year of their introduction.

year	transistors
1971	2,250
1972	2,500
1974	5,000
1978	29,000
1982	120,000
1985	275,000
1989	1,180,000
1993	3,100,000
1997	7,500,000
1999	24,000,000
2000	42,000,000
2002	220,000,000
2003	410,000,000



These numbers are available in the MATLAB file `ch8ex3.m`. If you run `[t,n] = ch8ex3.m`, then `t` is the first column of the table (introduction year) and `n` is the second column (number of transistors). The plot right-hand plot was produced by the MATLAB command `semilogy(t,n,'o')`).

The plot suggests that we can obtain a good fit with a function of the form

$$n(t) = \alpha^{t-t_0},$$

where t is the year, and n is the number of transistors. This is a straight line if we plot $n(t)$ on a logarithmic scale versus t on a linear scale. In this problem we use least-squares to estimate the parameters α and t_0.

Explain how you would use least-squares to find α and t_0 such that

$$n_i \approx \alpha^{t_i-t_0}, \quad i = 1, \ldots, 13,$$

and solve the least-squares problem in MATLAB. Compare your result with Moore's law, which states that the number of transistors per integrated circuit roughly doubles every two years.

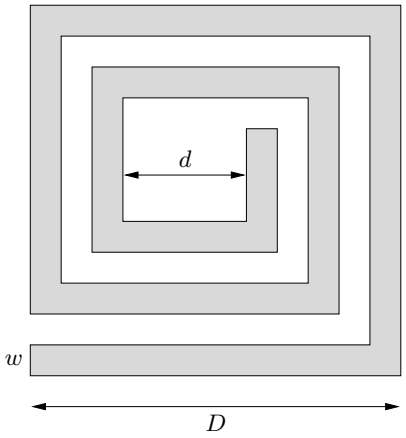
Remark. The MATLAB command to solve a least-squares problem

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

is `x = A\b`, *i.e.*, the same command as for solving a set of linear equations. In other words, the meaning of the backslash operator depends on the context. If A is a square matrix, then `A\b` solves the linear set of equations $Ax = b$; if A is rectangular with more rows than columns, it solves the least-squares problem.

8.4 The figure shows a planar spiral inductor, implemented in CMOS, for use in RF circuits. The inductor is characterized by four key parameters:

- n, the number of turns (which is a multiple of 1/4, but that needn't concern us)
- w, the width of the wire
- d, the inner diameter
- D, the outer diameter



The inductance L of such an inductor is a complicated function of the parameters n, w, d, and D. It can be found by solving Maxwell's equations, which takes considerable computer time, or by fabricating the inductor and measuring the inductance. In this problem you will develop a simple approximate inductance model of the form

$$\hat{L} = \alpha n^{\beta_1} w^{\beta_2} d^{\beta_3} D^{\beta_4},$$

where $\alpha, \beta_1, \beta_2, \beta_3, \beta_4 \in \mathbf{R}$ are constants that characterize the approximate model. (Since L is positive, we have $\alpha > 0$, but the constants $\beta_2, \ldots, \beta_4$ can be negative.) This simple approximate model, if accurate enough, can be used for design of planar spiral inductors.

The file `ch8ex4.m` contains data for 50 inductors, obtained from measurements. Download the file, and execute it in MATLAB using `[n,w,d,D,L] = ch8ex4`. This generates 5 vectors n, w, d, D, L of length 50. The ith elements of these vectors are the parameters n_i, w_i (in μm), d_i (in μm), D_i (in μm) and the inductance L_i (in nH) for inductor i.. Thus, for example, w_{13} gives the wire width of inductor 13.

Your task is to find α, $\beta_1, \ldots, \beta_4$ so that

$$\widehat{L}_i = \alpha n_i^{\beta_1} w_i^{\beta_2} d_i^{\beta_3} D_i^{\beta_4} \approx L_i \qquad \text{for } i = 1, \ldots, 50.$$

Your solution must include a clear description of how you found your parameters, as well as their actual numerical values.

Note that we have not specified the criterion that you use to judge the approximate model (*i.e.*, the fit between $\widehat{L}_i$ and L_i); we leave that to your judgment.

We can define the *percentage error* between $\widehat{L}_i$ and L_i as

$$e_i = 100 \frac{|\widehat{L}_i - L_i|}{L_i}.$$

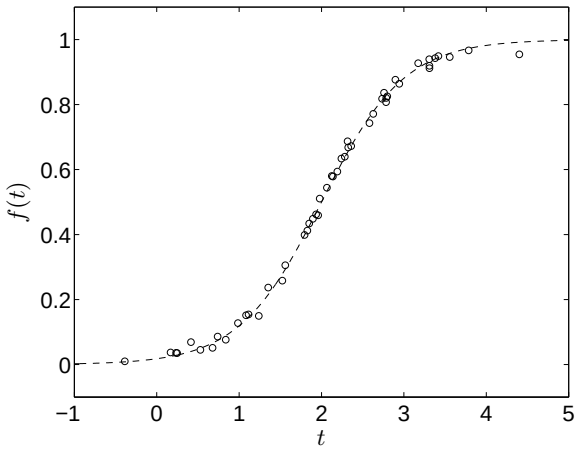
Find the average percentage error for the 50 inductors, *i.e.*, $(e_1 + \cdots + e_{50})/50$, for your model. (We are only asking you to find the average percentage error for your model; we do not require that your model minimize the average percentage error.)

Remark. For details on solving least-squares problems in MATLAB, see the remark at the end of exercise 8.3.

8.5 The figure shows $m = 50$ points (t_i, y_i) as circles. These points are well approximated by a function of the form

$$f(t) = \frac{e^{\alpha t + \beta}}{1 + e^{\alpha t + \beta}}.$$

(An example is shown in dashed line).



Formulate the following problem as a linear least-squares problem. Find values of the parameters α, β such that

$$\frac{e^{\alpha t_i+\beta}}{1+e^{\alpha t_i+\beta}} \approx y_i, \qquad i = 1, \ldots, m, \tag{8.9}$$

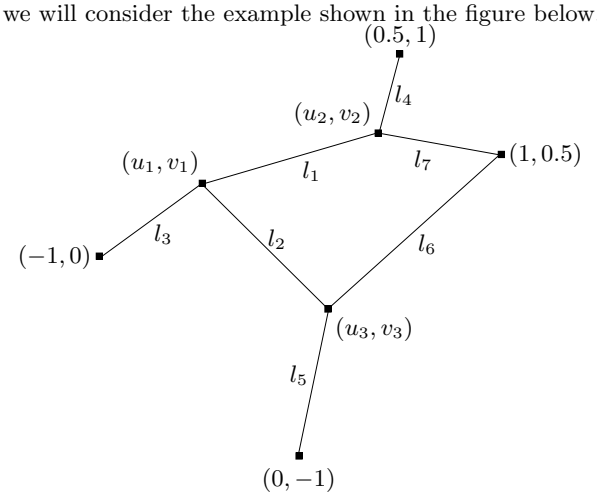
You can assume that $0 < y_i < 1$ for $i = 1, \ldots, m$.

Clearly state the error function you choose to measure the quality of the fit in (8.9), and the matrix A and the vector b of the least-squares problem.

8.6 We have N points in $\mathbf{R}^2$, and a list of pairs of points that must be connected by links. The positions of some of the N points are fixed; our task is to determine the positions of the remaining points. The objective is to place the points so that some measure of the total interconnection length of the links is minimized. As an example application, we can think of the points as locations of plants or warehouses of a company, and the links as the routes over which goods must be shipped. The goal is to find locations that minimize the total transportation cost. In another application, the points represent the position of modules or cells on an integrated circuit, and the links represent wires that connect pairs of cells. Here the goal might be to place the cells in such a way that the the total length of wire used to interconnect the cells is minimized.

The problem can be described in terms of a graph with N nodes, representing the N points. With each free node we associate a variable $(u_i, v_i) \in \mathbf{R}^2$, which represents its location or position.

In this problem we will consider the example shown in the figure below.



Here we have 3 free points with coordinates (u_1, v_1), (u_2, v_2), (u_3, v_3). We have 4 fixed points, with coordinates $(-1, 0)$, $(0.5, 1)$, $(0, -1)$, and $(1, 0.5)$. There are 7 links, with lengths l_1, l_2, ..., l_7. We are interested in finding the coordinates (u_1, v_1), (u_2, v_2) and (u_3, v_3) that minimize the total squared length

$$l_1^2 + l_2^2 + l_3^2 + l_4^2 + l_5^2 + l_6^2 + l_7^2.$$

(a) Formulate this problem as a least-squares problem

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

where the 6-vector x contains the six variables u_1, u_2, u_3, v_1, v_2, v_3. Give the coefficient matrix A and the vector b.

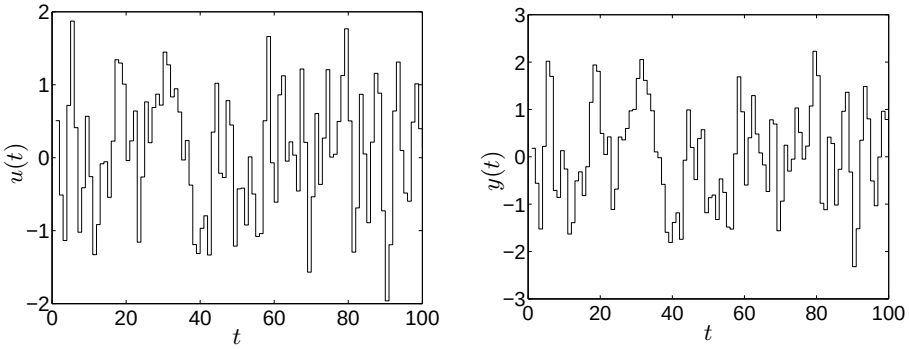
(b) Show that you can also obtain the optimal coordinates by solving two smaller least-squares problems

$$\text{minimize} \quad \|\bar{A}u - \bar{b}\|^2, \qquad \text{minimize} \quad \|\hat{A}v - \hat{b}\|^2,$$

where $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$. Give the coefficient matrices $\bar{A}$, $\hat{A}$ and the vectors $\bar{b}$ and $\hat{b}$. What is the relation between $\bar{A}$ and $\hat{A}$?

(c) Solve the least-squares problems derived in part (a) or (b) using MATLAB.

8.7 *Least-squares model fitting.* In this problem we use least-squares to fit several different types of models to a given set of input-output data. The data set consists of a scalar input sequence $u(1)$, $u(2)$, ..., $u(N)$, and a scalar output sequence $y(1)$, $y(2)$, ..., $y(N)$, with $N = 100$. The signals are shown in the following plots.



We will develop and compare seven different models that relate the signals u and y. The models range in complexity from a simple constant to a nonlinear dynamic model:

(a) constant model: $y(t) = \alpha$

(b) static linear: $y(t) = \beta u(t)$

(c) static affine: $y(t) = \alpha + \beta u(t)$

(d) static quadratic: $y(t) = \alpha + \beta u(t) + \gamma u(t)^2$

(e) linear, 2-tap: $y(t) = \beta_1 u(t) + \beta_2 u(t-1)$

(f) affine, 2-tap: $y(t) = \alpha + \beta_1 u(t) + \beta_2 u(t-1)$

(g) quadratic, 2-tap: $y(t) = \alpha + \beta_1 u(t) + \gamma_1 u(t)^2 + \beta_2 u(t-1) + \gamma_2 u(t-1)^2 + \delta u(t)u(t-1)$.

The first four models are *memoryless*. In a memoryless model the output at time t, *i.e.*, $y(t)$, depends only the input at time t, *i.e.*, $u(t)$. Another common term for such a model is *static*.

In a *dynamic model*, $y(t)$ depends on $u(s)$ for some $s \neq t$. Models (e), (f), and (g) are dynamic models, in which the current output depends on the current input and the previous input. Such models are said to have a *finite memory* of length one. Another term is 2-tap system (the taps refer to taps on a delay line).

Each of the models is specified by a number of parameters, *i.e.*, the scalars α, β, etc. You are asked to find least-squares estimates ($\hat{\alpha}$, $\hat{\beta}$, ...) for the parameters, *i.e.*, the values that minimize the sum-of-squares of the errors between predicted outputs and actual outputs. Your solutions should include:

- a clear description of the least-squares problems that you solve
- the computed values of the least-squares estimates of the parameters
- a plot of the predicted output $\hat{y}(t)$
- a plot of the residual $\hat{y}(t) - y(t)$
- the root-mean-square (RMS) residual, *i.e.*, the squareroot of the mean of the squared residuals.

For example, the affine 2-tap model (part (f)) depends on three parameters α, β_1, and β_2. The least-squares estimates $\hat{\alpha}$, $\hat{\beta}_1$, $\hat{\beta}_2$ are found by minimizing

$$\sum_{t=2}^{N} \left(y(t) - \alpha - \beta_1 u(t) - \beta_2 u(t-1)\right)^2 .$$

(Note that we start at $t = 2$ so $u(t-1)$ is defined). You are asked to formulate this as a least-squares problem, solve it to find $\hat{\alpha}$, $\hat{\beta}_1$, and $\hat{\beta}_2$, plot the predicted output

$$\hat{y}(t) = \hat{\alpha} + \hat{\beta}_1 u(t) + \hat{\beta}_2 u(t-1),$$

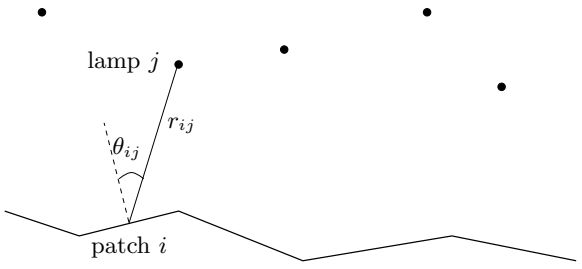
and the residual $r(t) = \hat{y}(t) - y(t)$, for $t = 2, \ldots, N$, and give the value of the RMS residual

$$R_{\mathrm{rms}} = \left(\frac{1}{N-1} \sum_{t=2}^{N} (y(t) - \hat{y}(t))^2 \right)^{1/2} .$$

The data for the problem are available from the class webpage in the m-file `ch8ex7.m`. The command is `[u,y] = ch8ex7`.

A final note: the sequences u, y are *not* generated by any of the models above. They are generated by a nonlinear recursion, with infinite (but rapidly fading) memory.

8.8 The figure shows an illumination system of n lamps illuminating m flat patches. The variables in the problem are the lamp powers $x_1, \ldots, x_n$, which can vary between 0 and 1.



The illumination intensity at (the midpoint of) patch i is denoted I_i. We will use a simple linear model for the illumination intensities I_i as a function of the lamp powers x_j: for $i = 1, \ldots, m$,

$$I_i = \sum_{j=1}^{n} a_{ij} x_j .$$

The matrix A (with coefficients a_{ij}) is available from the class webpage (see below), and was constructed as follows. We take

$$a_{ij} = r_{ij}^{-2} \max\{\cos\theta_{ij}, 0\},$$

where r_{ij} denotes the distance between lamp j and the midpoint of patch i, and θ_{ij} denotes the angle between the upward normal of patch i and the vector from the midpoint of patch i to lamp j, as shown in the figure. This model takes into account "self-shading" (*i.e.*, the fact that a patch is illuminated only by lamps in the halfspace it faces) but not shading of one patch caused by another. Of course we could use a more complex illumination model, including shading and even reflections. This just changes the matrix relating the lamp powers to the patch illumination levels.

The problem is to determine lamp powers that make the illumination levels I_i close to a given desired level $I_{\rm des}$. In other words, we want to choose the n-vector x such that

$$\sum_{j=1}^{n} a_{ij}x_j \approx I_{\rm des}, \quad i = 1, \ldots, m,$$

but we also have to observe the power limits $0 \leq x_j \leq 1$. This is an example of a *constrained optimization problem*. The objective is to achieve an illumination level that is as uniform as possible; the constraint is that the components of x must satisfy $0 \leq x_j \leq 1$. Finding the exact solution of this minimization problem requires specialized numerical techniques for constrained optimization. However, we can solve it *approximately* using least-squares.

In this problem we consider two approximate methods that are based on least-squares, and compare them for the data generated using `[A,Ides] = ch8ex8`, with the MATLAB `ch8ex8.m`. The elements of A are the coefficients a_{ij}. In this example we have $m = 11$, $n = 7$ so A is 11×7, and $I_{\rm des} = 2$.

(a) *Saturate the least-squares solution.* The first method is simply to ignore the bounds on the lamp powers. We solve the least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{m}\Big(\sum_{j=1}^{n} a_{ij}x_j - I_{\rm des}\Big)^2$$

ignoring the constraints $0 \leq x_j \leq 1$. If we are lucky, the solution will satisfy the bounds $0 \leq x_j \leq 1$, for $j = 1, \ldots, n$. If not, we replace x_j with zero if $x_j < 0$ and with one if $x_j > 1$.

Apply this method to the problem data generated by `ch8ex8.m`, and calculate the resulting value of the cost function $\sum_{i=1}^{m}(I_i - I_{\rm des})^2$.

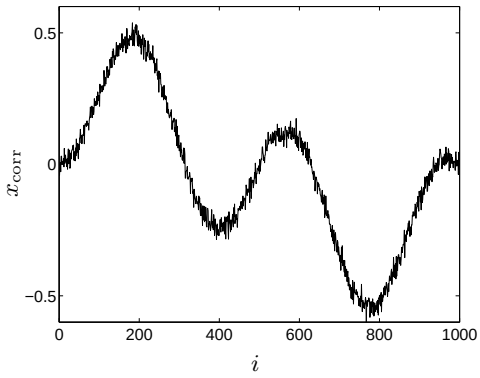
(b) *Weighted least-squares.* The second method is to solve the problem

$$\text{minimize} \quad \sum_{i=1}^{m}\Big(\sum_{j=1}^{n} a_{ij}x_j - I_{\rm des}\Big)^2 + \mu\sum_{j=1}^{n}(x_j - 0.5)^2,$$

where the constant $\mu \geq 0$ is used to attach a cost to the deviation of the powers from the value 0.5, which lies in the middle of the power limits. For $\mu = 0$, this is the same least-squares problem as in part (a). If we take μ large enough, the solution of this problem will satisfy $0 \leq x_j \leq 1$.

Formulate this problem as a least-squares problem in the variables x, and solve it for $\mu = 1$, $\mu = 2$, $\mu = 3$, etc., until you find a value of μ such that all components of the solution x satisfy $0 \leq x_j \leq 1$. For that solution x, calculate the cost function $\sum_{i=1}^{m}(I_i - I_{\rm des})^2$ and compare with the value you obtained in part (a).

8.9 *De-noising using least-squares.* The figure shows a signal of length 1000, corrupted with noise. We are asked to estimate the original signal. This is called signal reconstruction, or de-noising, or smoothing. In this problem we apply a smoothing method based on least-squares.



We will represent the corrupted signal as a vector $x_{\rm cor}$ of size 1000. (The values can be obtained as `xcor = ch8ex9` using the file `ch8ex9.m`.) The estimated signal (*i.e.*, the variable in the problem) will be represented as a vector $\hat{x}$ of size 1000.

The idea of the method is as follows. We assume that the noise in the signal is the small and rapidly varying component. To reconstruct the signal, we decompose $x_{\rm cor}$ in two parts

$$x_{\rm cor} = \hat{x} + v$$

where v is small and rapidly varying, and $\hat{x}$ is close to $x_{\rm cor}$ ($\hat{x} \approx x_{\rm cor}$) and slowly varying ($\hat{x}_{i+1} \approx \hat{x}_i$). We can achieve such a decomposition by choosing $\hat{x}$ as the solution of the least-squares problem

$$\text{minimize} \quad \|x - x_{\rm cor}\|^2 + \mu \textstyle\sum_{i=1}^{999} (x_{i+1} - x_i)^2, \tag{8.10}$$

where μ is a positive constant. The first term $\|x - x_{\rm cor}\|^2$ measures how much x deviates from $x_{\rm cor}$. The second term, $\sum_{i=1}^{999}(x_{i+1} - x_i)^2$, penalizes rapid changes of the signal between two samples. By minimizing a weighted sum of both terms, we obtain an estimate $\hat{x}$ that is close to $x_{\rm cor}$ (*i.e.*, has a small value of $\|\hat{x} - x_{\rm cor}\|^2$) and varies slowly (*i.e.*, has a small value of $\sum_{i=1}^{999}(\hat{x}_{i+1} - \hat{x}_i)^2$). The parameter μ is used to adjust the relative weight of both terms.

Problem (8.10) is a least-squares problem, because it can be expressed as

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

where

$$A = \begin{bmatrix} I \\ \sqrt{\mu}\, D \end{bmatrix}, \qquad b = \begin{bmatrix} x_{\rm cor} \\ 0 \end{bmatrix},$$

and D is a 999×1000-matrix defined as

$$D = \begin{bmatrix} -1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 1 \end{bmatrix}.$$

The matrix A is quite large (1999×1000), but also very sparse, so we will solve the least-squares problem using the Cholesky factorization method. You should verify that the normal equations are given by

$$(I + \mu D^T D)x = x_{\text{cor}}. \tag{8.11}$$

MATLAB and Octave provide special routines for solving sparse linear equations, and they are used as follows. There are two types of matrices: full (or dense) and sparse. If you define a matrix, it is considered full by default, unless you specify that it is sparse. You can convert a full matrix to sparse format using the command `A = sparse(A)`, and a sparse matrix to full format using the command `A = full(A)`.

When you type `x = A\b` where A is $n \times n$, MATLAB chooses different algorithms depending on the type of A. If A is full it uses the standard method for general matrices (LU or Cholesky factorization, depending on whether A is symmetric positive definite or not). If A is sparse, it uses an LU or Cholesky factorization algorithm that takes advantage of sparsity. In our application, the matrix $I + \mu D^T D$ is sparse (in fact tridiagonal), so if we make sure to define it as a sparse matrix, the normal equations will be solved much more quickly than if we ignore the sparsity.

The command to create a sparse zero matrix of dimension $m \times n$ is `A = sparse(m,n)`. The command `A = speye(n)` creates a sparse $n \times n$-identity matrix. If you add or multiply sparse matrices, the result is automatically considered sparse.

This means you can solve the normal equations (8.11) by the following MATLAB code (assuming μ and x_{cor} are defined):

```
D = sparse(999,1000);
D(:,1:999) =  -speye(999);
D(:,2:1000) =  D(:,2:1000) + speye(999);
xhat = (speye(1000) + mu*D'*D) \ xcor;
```

Solve the least-squares problem (8.10) with the vector x_{cor} defined in `ch8ex9.m`, for three values of μ: $\mu = 1$, $\mu = 100$, and $\mu = 10000$. Plot the three reconstructed signals $\hat{x}$. Discuss the effect of μ on the quality of the estimate $\hat{x}$.

8.10 Suppose you are given m ($m \geq 2$) straight lines

$$L_i = \{p_i + t_i q_i \mid t_i \in \mathbf{R}\}, \quad i = 1, \ldots, m$$

in $\mathbf{R}^n$. Each line is defined by two n-vectors p_i, q_i. The vector p_i is a point on the line; the vector q_i specifies the direction. We assume that the vectors q_i are normalized ($\|q_i\| = 1$) and that at least two of them are linearly independent. (In other words, the vectors q_i are not all scalar multiples of the same vector, so the lines are not all parallel.) We denote by

$$d_i(y) = \min_{u_i \in L_i} \|y - u_i\| = \min_{t_i} \|y - p_i - t_i q_i\|$$

the distance of a point y to the line L_i.

Express the following problem as a linear least-squares problem. Find the point $y \in \mathbf{R}^n$ that minimizes the sum of its squared distances to the m lines, *i.e.*, find the solution of the optimization problem

$$\text{minimize} \quad \sum_{i=1}^{m} d_i(y)^2$$

with variable y. Express the least-squares problem in the standard form

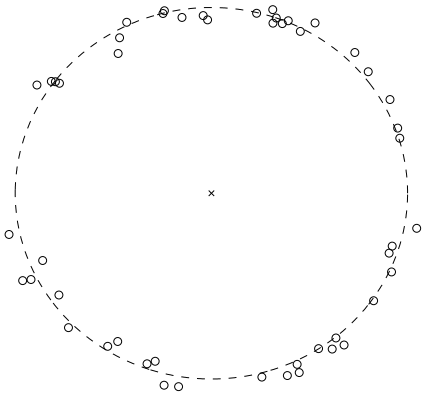
$$\text{minimize} \quad \|Ax - b\|^2$$

with a left-invertible (zero nullspace) matrix A.

(a) Clearly state what the variables x in the least-squares problem are and how A and b are defined.

(b) Explain why A has a zero nullspace.

8.11 In this problem we use least-squares to fit a circle to given points (u_i, v_i) in a plane, as shown in the figure.



We use (u_c, v_c) to denote the center of the circle and R for its radius. A point (u, v) is on the circle if $(u - u_c)^2 + (v - v_c)^2 = R^2$. We can therefore formulate the fitting problem as

$$\text{minimize} \quad \sum_{i=1}^{m} \left((u_i - u_c)^2 + (v_i - v_c)^2 - R^2 \right)^2$$

with variables u_c, v_c, R.
Show that this can be written as a linear least-squares problem if we make a change of variables and use as variables u_c, v_c, and $w = u_c^2 + v_c^2 - R^2$.

(a) Define A, b, and x in the equivalent linear least-squares formulation.

(b) Show that the optimal solution u_c, v_c, w of the least-squares problem satisfies $u_c^2 + v_c^2 - w \geq 0$. (This is necessary to compute $R = \sqrt{u_c^2 + v_c^2 - w}$ from the result u_c, v_c, w.)

Test your formulation on the problem data in the file `ch8ex11.m` on the course website. The commands

```
[u,v] = ch8ex11
plot(u, v, 'o');
axis square
```

will create a plot of the $m = 50$ points (u_i, v_i) in the figure. The following code plots the 50 points and the computed circle.

```
t = linspace(0, 2*pi, 1000);
plot(u, v, 'o',  R * cos(t) + uc, R * sin(t) + vc, '-');
axis square
```

(assuming your MATLAB variables are called `uc`, `vc`, and `R`).

The solution of a least-squares problem

8.12 Let A be a left-invertible $m \times n$-matrix.

(a) Show that the $(m + n) \times (m + n)$ matrix

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}$$

is nonsingular.

(b) Show that the solution $\bar{x}$, $\bar{y}$ of the set of linear equations

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

is given by $\bar{x} = b - Ax_{\rm ls}$ and $\bar{y} = x_{\rm ls}$ where $x_{\rm ls}$ is the solution of the least-squares problem

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

8.13 Consider the set of $p + q$ linear equations in $p + q$ variables

$$\begin{bmatrix} I & A \\ A^T & -I \end{bmatrix} \begin{bmatrix} \bar{y} \\ \bar{x} \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}.$$

The $p \times q$-matrix A, the p-vector b, and the q-vector c are given. The variables are the q-vector $\bar{x}$ and the p-vector $\bar{y}$.

(a) Show that the coefficient matrix

$$\begin{bmatrix} I & A \\ A^T & -I \end{bmatrix}$$

is nonsingular, regardless of the dimensions of A.

(b) From part (a) we know that the solution $\bar{x}$, $\bar{y}$ is unique. Show that $\bar{x}$ minimizes

$$\|Ax - b\|^2 + \|x + c\|^2.$$

8.14 Define $\hat{x}$ as the solution of the optimization problem

$$\text{minimize} \quad \|Ax - b\|^2 + \|x\|^2.$$

A is an $m \times n$ matrix and b is an m-vector. We make no assumptions about the dimensions and properties of A.

(a) Show that $\hat{x} = (A^T A + I)^{-1} A^T b$.

(b) Show that $\hat{x} = A^T (AA^T + I)^{-1} b$. (*Hint.* Show that $A^T (AA^T + I)^{-1} = (A^T A + I)^{-1} A^T$.)

(c) Describe an efficient algorithm for computing $\hat{x}$ using a Cholesky factorization. Distinguish two cases: $m \geq n$ and $m \leq n$. Carefully explain the different steps in your algorithm, the complexity (number of flops for large m, n) of each step, and the total complexity.

Chapter 9

QR factorization

The Cholesky factorization method for solving least-squares problems is often used in practice, but it is not the most accurate method for solving a least-squares problem, for reasons that we will clarify later. The recommended method is based on the QR factorization of A.

9.1 Orthogonal matrices

Recall that a matrix Q is *orthogonal* if $Q^TQ = I$. The following properties of an orthogonal $m \times n$-matrix Q are useful (see chapter 2 and section 3.5).

- Q is left-invertible. Its transpose is a left inverse.
- Q has a zero nullspace. This is true for all left-invertible matrices. It can also be verified directly: $Qx = 0$ implies $x = Q^TQx = 0$ because $Q^TQ = I$.
- $m \geq n$. This is true for all left-invertible matrices.
- $\|Qx\| = \|x\|$ for all x and $(Qx)^T(Qy) = x^Ty$ for all x and y. In other words multiplying vectors with an orthogonal matrix preserves norms and inner products. As a consequence, it also preserves the angle between vectors:

  $$\cos\theta = \frac{(Qx)^T(Qy)}{\|Qx\|\,\|Qy\|} = \frac{x^Ty}{\|x\|\,\|y\|} = \cos\gamma,$$

  if θ is the angle between Qx and Qy, and γ is the angle between x and y.
- $\|Q\| = 1$. This follows from the definition of matrix norm, and the previous property:

  $$\|Q\| = \max_{x\neq 0} \frac{\|Qx\|}{\|x\|} = \max_{x\neq 0} \frac{\|x\|}{\|x\|} = 1.$$

- If $m = n$, then $Q^{-1} = Q^T$ and we also have $QQ^T = I$.

It is important to keep in mind that the last property ($QQ^T = I$) only holds for *square* orthogonal matrices. If $m > n$, then $QQ^T \neq I$.

9.2 Definition

A left-invertible $m \times n$-matrix A can be factored as

$$A = QR$$

where Q is an $m \times n$-orthogonal matrix and R is an $n \times n$-upper triangular matrix with positive diagonal elements. This is called the *QR factorization* of A. In section 9.4 we will describe an algorithm that computes the QR factorization in $2mn^2$ flops.

Example One can verify that the matrix A given in (8.8) can be factored as $A = QR$ with

$$Q = \begin{bmatrix} 3/5 & 0 & 4/5 \\ 4/5 & 0 & -3/5 \\ 0 & 4/5 & 0 \\ 0 & -3/5 & 0 \end{bmatrix}, \qquad R = \begin{bmatrix} 1 & -2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 5 \end{bmatrix}, \tag{9.1}$$

and that $Q^TQ = I$.

9.3 Solving least-squares problems by QR factorization

The QR factorization provides an alternative method for solving the normal equations introduced in section 8.4. If $A = QR$ and $Q^TQ = I$, then

$$A^TA = (QR)^T(QR) = R^TQ^TQR = R^TR,$$

so the normal equations (8.7) reduce to

$$R^TRx = R^TQ^Tb.$$

The matrix R is invertible (it is upper triangular with positive diagonal elements), so we can multiply with R^{-T} on both sides, and obtain

$$Rx = Q^Tb.$$

This suggests the following method for solving the least-squares problem.

Algorithm 9.1. Solving least-squares problems by QR factorization.

given a left-invertible $m \times n$-matrix A and an m-vector b.

1. Compute the QR factorization $A = QR$.
2. Compute $d = Q^Tb$.
3. Solve $Rx = d$ by back substitution.

The cost of step 1 (QR factorization) is $2mn^2$. Step 2 costs $2mn$, and step 3 costs n^2, so the total cost is $2mn^2 + 2mn + n^2$ or roughly

$$2mn^2.$$

We see that the QR factorization method is always slower than the Cholesky factorization method (which costs $mn^2 + (1/3)n^3$). It is about twice as slow if $m \gg n$.

Example As an example, we can solve the least-squares problem with A and b given in (8.8), using the QR factorization of A given in (9.1). We have

$$Q^T b = \begin{bmatrix} 3/5 & 4/5 & 0 & 0 \\ 0 & 0 & 4/5 & -3/5 \\ 4/5 & -3/5 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7/5 \\ 1/5 \\ 1/5 \end{bmatrix}.$$

To find the solution, we solve $Rx = Q^T b$, *i.e.*,

$$\begin{bmatrix} 1 & -2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7/5 \\ 1/5 \\ 1/5 \end{bmatrix}$$

by back substitution. The solution is $x = (41/25, 4/25, 1/25)$.

9.4 Computing the QR factorization

We now turn to the question of computing the QR factorization of a left-invertible $m \times n$-matrix A.

We start by partitioning A, Q, R as follows

$$A = \begin{bmatrix} a_1 & A_2 \end{bmatrix}, \qquad Q = \begin{bmatrix} q_1 & Q_2 \end{bmatrix}, \qquad R = \begin{bmatrix} r_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \tag{9.2}$$

where a_1 is $m \times 1$, A_2 is $m \times (n-1)$, q_1 is $m \times 1$, Q_2 is $m \times (n-1)$, r_{11} is a scalar, R_{12} is $1 \times (n-1)$, and R_{22} is $(n-1) \times (n-1)$. We want Q to be orthogonal, *i.e.*,

$$Q^T Q = \begin{bmatrix} q_1^T \\ Q_2^T \end{bmatrix} \begin{bmatrix} q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} q_1^T q_1 & q_1^T Q_2 \\ Q_2^T q_1 & Q_2^T Q_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & I \end{bmatrix},$$

which gives the following conditions on q_1 and Q_2:

$$q_1^T q_1 = 1, \qquad q_1^T Q_2 = 0, \qquad Q_2^T Q_2 = I. \tag{9.3}$$

In addition we want R upper triangular with positive diagonal elements, so $r_{11} > 0$, and R_{22} is upper triangular with positive diagonal elements.

To determine Q and R we write the identity $A = QR$ in terms of the partitioned matrices

$$\begin{aligned} \begin{bmatrix} a_1 & A_2 \end{bmatrix} &= \begin{bmatrix} q_1 & Q_2 \end{bmatrix} \begin{bmatrix} r_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \\ &= \begin{bmatrix} q_1 r_{11} & q_1 R_{12} + Q_2 R_{22} \end{bmatrix}. \end{aligned}$$

Comparing the first columns we see that $a_1 = q_1 r_{11}$. The vector q_1 must have unit norm (see the first condition in (9.3)), and r_{11} must be positive. We therefore choose

$$r_{11} = \|a_1\|, \qquad q_1 = (1/r_{11})a_1.$$

The next step is to determine R_{12}. From (9.4), we have

$$A_2 = q_1 R_{12} + Q_2 R_{22}.$$

Multiplying both sides of the equality on the left with q_1^T and using the fact that $q_1^T q_1 = 1$ and $q_1^T Q_2 = 0$ (from (9.3)), we obtain

$$q_1^T A_2 = q_1^T q_1 R_{12} + q_1^T Q_2 R_{22} = R_{12}$$

so we can compute R_{12} as $R_{12} = q_1^T A_2$. Finally, we compute Q_2, R_{22} from the identity

$$A_2 - q_1 R_{12} = Q_2 R_{22}. \tag{9.4}$$

The matrix on the left is known, because we have already computed q_1 and R_{12}. The matrix Q_2 on the right-hand side must be orthogonal ($Q_2^T Q_2 = I$), and R_{22} must be upper triangular with positive diagonal. We can therefore interpret (9.4) as the QR factorization of a matrix of size $m \times (n-1)$.

Continuing recursively, we arrive at the QR factorization of an $m \times 1$ matrix, which is straightforward: if A is $m \times 1$, then it can be factored as $A = QR$ with $Q = (1/\|A\|)A$ and $R = \|A\|$.

This algorithm is called the *modified Gram-Schmidt method*. Referring to the notation of (9.2) we can summarize the algorithm as follows.

Algorithm 9.2. Modified Gram-Schmidt method for QR factorization.

given a left-invertible $m \times n$-matrix A.

1. $r_{11} = \|a_1\|$.
2. $q_1 = (1/r_{11})a_1$.
3. $R_{12} = q_1^T A_2$.
4. Compute the QR factorization $A_2 - q_1 R_{12} = Q_2 R_{22}$.

It can be shown that the total cost is $2mn^2$.

In step 1, we can be assured that $\|a_1\| \neq 0$ because A has a zero nullspace. (If the first column of A were zero, the unit vector e_1 would be in the nullspace.) We also have to verify that the matrix $A_2 - q_1 R_{12}$ in step 4 is left-invertible if A is left-invertible. To show this, suppose $x \neq 0$. Then

$$\begin{aligned}
(A_2 - q_1 R_{12})x &= A_2 x - (1/r_{11})a_1 R_{12} x \\
&= \begin{bmatrix} a_1 & A_2 \end{bmatrix} \begin{bmatrix} -(1/r_{11})R_{12}x \\ x \end{bmatrix} \\
&\neq 0,
\end{aligned}$$

because $A = \begin{bmatrix} a_1 & A_2 \end{bmatrix}$ has a zero nullspace.

Example We apply the modified Gram-Schmidt method to the matrix

$$A = (1/5)\begin{bmatrix} 3 & -6 & 26 \\ 4 & -8 & -7 \\ 0 & 4 & 4 \\ 0 & -3 & -3 \end{bmatrix}.$$

We will denote the three columns as a_1, a_2, a_3, and factor A as

$$\begin{aligned} \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} &= \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix} \\ &= \begin{bmatrix} q_1 r_{11} & q_1 r_{12} + q_2 r_{22} & q_1 r_{13} + q_2 r_{23} + q_3 r_{33} \end{bmatrix}, \end{aligned}$$

where the vectors q_i are mutually orthogonal and unit norm, and the diagonal elements r_{ii} are positive.

We start with q_1 and r_{11}, which must satisfy $a_1 = q_1 r_{11}$ with $\|q_1\| = 1$. Therefore

$$r_{11} = \|a_1\| = 1, \qquad q_1 = (1/r_{11})a_1 = \begin{bmatrix} 3/5 \\ 4/5 \\ 0 \\ 0 \end{bmatrix}.$$

Next we find r_{12} and r_{13} by multiplying

$$\begin{bmatrix} a_2 & a_3 \end{bmatrix} = \begin{bmatrix} q_1 r_{12} + q_2 r_{22} & q_1 r_{13} + q_2 r_{23} + q_3 r_{33} \end{bmatrix}$$

on the left with q_1^T and using $q_1^T q_1 = 1$, $q_1^T q_2 = 0$, $q_1^T q_3 = 0$:

$$\begin{aligned} q_1^T \begin{bmatrix} a_2 & a_3 \end{bmatrix} &= q_1^T \begin{bmatrix} q_1 r_{12} + q_2 r_{22} & q_1 r_{13} + q_2 r_{23} + q_3 r_{33} \end{bmatrix} \\ &= \begin{bmatrix} r_{12} & r_{13} \end{bmatrix}. \end{aligned}$$

We have just computed q_1, so we can evaluate the matrix on the left, and find $r_{12} = -2$, $r_{13} = 2$. This concludes steps 1–3 in the algorithm.

The next step is to factor the matrix

$$\begin{bmatrix} \tilde{a}_2 & \tilde{a}_3 \end{bmatrix} = \begin{bmatrix} a_2 - q_1 r_{12} & a_3 - q_1 r_{13} \end{bmatrix} = \begin{bmatrix} 0 & 4 \\ 0 & -3 \\ 4/5 & 4/5 \\ -3/5 & -3/5 \end{bmatrix}$$

as

$$\begin{aligned} \begin{bmatrix} \tilde{a}_2 & \tilde{a}_3 \end{bmatrix} &= \begin{bmatrix} q_2 & q_3 \end{bmatrix} \begin{bmatrix} r_{22} & r_{23} \\ 0 & r_{33} \end{bmatrix} \\ &= \begin{bmatrix} q_2 r_{22} & q_2 r_{23} + q_3 r_{33} \end{bmatrix}. \end{aligned} \tag{9.5}$$

We start with q_2 and r_{22}. We want $\tilde{a}_2 = q_2 r_{22}$ with $\|q_2\| = 1$, so we take

$$r_{22} = \|\tilde{a}_2\| = 1, \qquad q_2 = (1/r_{22})\tilde{a}_2 = \begin{bmatrix} 0 \\ 0 \\ 4/5 \\ -3/5 \end{bmatrix}.$$

Next, we determine r_{23} from the second column in (9.5),

$$\tilde{a}_3 = q_2 r_{23} + q_3 r_{33}.$$

If we multiply on the left with q_2^T and use the fact that $q_2^T q_2 = 1$, $q_2^T q_3 = 0$, we find

$$q_2^T \tilde{a}_3 = q_2^T (q_2 r_{23} + q_3 r_{33}) = r_{23}.$$

We know q_2 and $\tilde{a}_3$, so we can evaluate the inner product on the left-hand side, which gives $r_{23} = 1$.

It remains to determine q_3 and r_{33}. At this point we know q_2 and r_{23}, so we can evaluate $\tilde{a}_3 - q_2 r_{23}$, and from the last column in (9.5),

$$\tilde{a}_3 - q_2 r_{23} = \begin{bmatrix} 4 \\ -3 \\ 0 \\ 0 \end{bmatrix} = q_3 r_{33},$$

with $\|q_3\| = 1$. Hence $r_{33} = 5$ and

$$q_3 = \begin{bmatrix} 4/5 \\ -3/5 \\ 0 \\ 0 \end{bmatrix}.$$

Putting everything together, the factorization of A is

$$(1/5) \begin{bmatrix} 3 & -6 & 26 \\ 4 & -8 & -7 \\ 0 & 4 & 4 \\ 0 & -3 & -3 \end{bmatrix} = \begin{bmatrix} 3/5 & 0 & 4/5 \\ 4/5 & 0 & -3/5 \\ 0 & 4/5 & 0 \\ 0 & -3/5 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 5 \end{bmatrix}.$$

9.5 Comparison with Cholesky factorization method

We conclude this chapter with a comparison of the two methods for solving least-squares problems.

Accuracy The Cholesky factorization method for solving least-squares problems is less accurate than the QR factorization method, especially when the condition number of the matrix $A^T A$ is high. An example will illustrate the difference. We take

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \\ 0 & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 10^{-5} \\ 1 \end{bmatrix}.$$

The normal equations are

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 + 10^{-10} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 10^{-10} \end{bmatrix}, \tag{9.6}$$

and it is easily verified that the solution is $x_1 = 1$, $x_2 = 1$. We will solve the problem using both methods, but introduce small errors by rounding the intermediate results to eight significant decimal digits.

We first consider the Cholesky factorization method. Rounding the elements of $A^T A$ and $A^T b$ in (9.6) to eight digits yields

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 10^{-10} \end{bmatrix}.$$

(Only one element changes: $1 + 10^{-10}$ is replaced by 1.) This set of equations is unsolvable because the coefficient matrix is singular, so the Cholesky factorization method fails.

In the QR factorization method we start by factoring A as

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \end{bmatrix}.$$

Rounding to eight decimal digits does not change the values of Q and R. We then form the equations $Rx = Q^T b$:

$$\begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 10^{-5} \end{bmatrix}.$$

Rounding to eight digits again does not change any of the values. We solve the equations by backward substitution and obtain the solution $x_1 = 1$, $x_2 = 1$.

In this example, the QR factorization method finds the correct solution, while the Cholesky factorization method fails.

Efficiency For dense matrices, the cost of the Cholesky factorization method is $mn^2 + (1/3)n^3$, while the cost of the QR factorization method is $2mn^2$. The QR factorization method is slower by a factor of at most 2 (if $m \gg n$). For small and medium-size problems, the factor of two does not outweigh the difference in accuracy, and the QR factorization is the recommended method.

When A is large and sparse, the difference in efficiency can be much larger than a factor of two. As we have mentioned in section 7.8, there exist very efficient methods for the Cholesky factorization of a sparse positive definite matrix. When A is sparse, then usually $A^T A$ is sparse (and can be calculated in much less than mn^2 operations), and we can solve the normal equations using a sparse Cholesky factorization (at a cost much less than $(1/3)n^3$).

Exploiting sparsity in the QR factorization method is much more difficult. The matrix Q is usually quite dense, even when the matrix A is sparse. This makes the QR method more expensive for large sparse least-squares problems. The Cholesky factorization method is therefore widely used when A is large and sparse, despite its lower accuracy.

Exercises

9.1 What is the QR factorization of the matrix

$$A = \begin{bmatrix} 2 & 8 & 13 \\ 4 & 7 & -7 \\ 4 & -2 & -13 \end{bmatrix} ?$$

You can use MATLAB to check your answer, but you must provide the details of all intermediate steps on paper.

9.2 Explain how you can solve the following problems using the QR factorization.

(a) Find the vector x that minimizes

$$\|Ax - b_1\|^2 + \|Ax - b_2\|^2.$$

The problem data are the $m \times n$-matrix A and two m-vectors b_1 and b_2. The matrix A is left-invertible. If you know several methods, give the most efficient one.

(b) Find x_1 and x_2 that minimize

$$\|Ax_1 - b_1\|^2 + \|Ax_2 - b_2\|^2.$$

The problem data are the $m \times n$-matrix A, and the m-vectors b_1 and b_2. A is left-invertible.

9.3 *Cholesky factorization versus QR factorization.* In this problem we compare the accuracy of the two methods for solving a least-squares problem

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

We take

$$A = \begin{bmatrix} 1 & 1 \\ 10^{-k} & 0 \\ 0 & 10^{-k} \end{bmatrix}, \qquad b = \begin{bmatrix} -10^{-k} \\ 1 + 10^{-k} \\ 1 - 10^{-k} \end{bmatrix},$$

for $k = 6$, $k = 7$ and $k = 8$.

(a) Write the normal equations, and solve them analytically (*i.e.*, on paper, without using MATLAB).

(b) Solve the least-squares problem in MATLAB, for $k = 6$, $k = 7$ and $k = 8$, using the recommended method `x = A\b`. This method is based on the QR factorization.

(c) Repeat part (b), using the Cholesky factorization method, *i.e.*, `x = (A'*A)\(A'*b)`. (We assume that MATLAB recognizes that A^TA is symmetric positive definite, and uses the Cholesky factorization to solve $A^TAx = A^Tb$). Compare the results of this method with the results of parts (a) and (b).

Remark. Type `format long` to make MATLAB display more than five digits.

9.4 Suppose $\hat{x}$ is the solution of the least-squares problem

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

where A is a left-invertible $m \times n$ matrix, and b is an m-vector.

(a) Show that the solution of the problem

$$\text{minimize} \quad \|Ay - b\|^2 + (c^Ty - d)^2$$

with variable y (where c is an n-vector, and d is a scalar) is given by

$$\hat{y} = \hat{x} + \frac{d - c^T\hat{x}}{1 + c^T(A^TA)^{-1}c}(A^TA)^{-1}c.$$

(b) Describe an efficient method for computing $\hat{x}$ and $\hat{y}$, given A, b, c and d, using the QR factorization of A. Clearly describe the different steps in your algorithm. Give a flop count for each step and a total flop count. In your total flop count, include all terms that are cubic (n^3, mn^2, m^2n, m^3) and quadratic (m^2, mn, n^2).
If you know several methods, give the most efficient one.

9.5 Let $\hat{x}$ and $\hat{y}$ be the solutions of the least-squares problems

$$\text{minimize} \quad \|Ax-b\|^2, \qquad \text{minimize} \quad \|Ay-c\|^2$$

where A is a left-invertible $m \times n$-matrix, and b and c are m-vectors. We assume that $A\hat{x} \neq b$.

(a) Show that the $m \times (n+1)$ matrix $\begin{bmatrix} A & b \end{bmatrix}$ is left-invertible.

(b) Show that the solution of the least-squares problem

$$\text{minimize} \quad \left\| \begin{bmatrix} A & b \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - c \right\|^2,$$

with variables $u \in \mathbf{R}^n$, $v \in \mathbf{R}$, is given by

$$\hat{u} = \hat{y} - \frac{b^Tc - b^TA\hat{y}}{b^Tb - b^TA\hat{x}}\hat{x}, \qquad \hat{v} = \frac{b^Tc - b^TA\hat{y}}{b^Tb - b^TA\hat{x}}.$$

(c) Describe an efficient method for computing $\hat{x}$, $\hat{y}$, $\hat{u}$, $\hat{v}$, given A, b, c, using the QR factorization of A. Clearly describe the different steps in your algorithm. Give a flop count for each step and a total flop count. In the total flop count, include all terms that are cubic (n^3, mn^2, m^2n, m^3) and quadratic (m^2, mn, n^2). If you know several methods, give the most efficient one (least number of flops for large m and n).

9.6 Let A be a left-invertible $m \times n$-matrix, and b an m-vector not in the range of A.

(a) Explain why the QR factorization

$$\begin{bmatrix} A & b \end{bmatrix} = QR$$

exists.

(b) Suppose we partition the matrices in the QR factorization of part 1 as

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}, \qquad R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},$$

where Q_1 is $m \times n$, Q_2 is $m \times 1$, R_{11} is $n \times n$, R_{12} is $n \times 1$ and R_{22} is a scalar. Show that $x_{\rm ls} = R_{11}^{-1}R_{12}$ is the solution of the least-squares problem

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

and that $R_{22} = \|Ax_{\rm ls} - b\|$.

9.7 An $m \times n$-matrix A is given in factored form

$$A = UDV^T$$

where U is $m \times n$ and orthogonal, D is $n \times n$ and diagonal with nonzero diagonal elements, and V is $n \times n$ and orthogonal. Describe an efficient method for solving the least-squares problem

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

'Efficient' here means that the complexity is substantially less than the complexity of the standard method based on the QR factorization. What is the cost of your algorithm (number of flops for large m and n)? (Note: we assume that U, D, V are given; you are not asked to include the cost of computing these matrices in the complexity analysis.)

Chapter 10

Least-norm problems

We now consider underdetermined sets of linear equations

$$Ax = b$$

where A is $m \times n$ with $m < n$. An underdetermined set of equations usually has infinitely many solutions. We can use this freedom to choose a solution that is more interesting than the others in some sense. The most common choice is to use the solution with the smallest norm $\|x\|$.

10.1 Definition

We use the notation

$$\begin{array}{ll} \text{minimize} & \|x\|^2 \\ \text{subject to} & Ax = b \end{array}$$

to denote the problem of finding an $\hat{x}$ that satisfies $A\hat{x} = b$, and $\|\hat{x}\|^2 \leq \|x\|^2$ for all x that satisfy $Ax = b$. In other words, $\hat{x}$ is the solution of the equations $Ax = b$ with the smallest norm. This is called a *least-norm problem* or *minimum-norm* problem.

Example We take

$$A = \begin{bmatrix} 1 & -1 & 2 \\ 1 & 1 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The corresponding least-norm problem is

$$\begin{array}{ll} \text{minimize} & x_1^2 + x_2^2 + x_3^2 \\ \text{subject to} & x_1 - x_2 + 2x_3 = 1 \\ & x_1 + x_2 - x_3 = 0. \end{array}$$

We can solve this particular problem as follows. (Systematic methods will be discussed later.) Using the second equation, we express x_3 in terms of x_1 and x_2

as $x_3 = x_1 + x_2$. Substituting in the first equation yields

$$x_1 - x_2 + 2(x_1 + x_2) = 3x_1 + x_2 = 1.$$

We eliminate x_2 from this equation,

$$x_2 = 1 - 3x_1.$$

This leaves us with one remaining variable x_1, which we have to determine by minimizing

$$\begin{aligned} x_1^2 + x_2^2 + x_3^2 &= x_1^2 + (1 - 3x_1)^2 + (x_1 + 1 - 3x_1)^2 \\ &= x_1^2 + (1 - 3x_1)^2 + (1 - 2x_1)^2. \end{aligned}$$

Setting to zero the derivative with respect to x_1 gives

$$2x_1 - 6(1 - 3x_1) - 4(1 - 2x_1) = 0.$$

The solution is $x_1 = 5/14$, $x_2 = 1 - 3x_1 = -1/14$, and $x_3 = x_1 + x_2 = 2/7$.

10.2 Solution of a least-norm problem

The following result provides the counterpart of the normal equations of a least-squares problem. If A is $m \times n$ and right-invertible, then the solution of the least-norm problem

$$\begin{array}{ll} \text{minimize} & \|x\|^2 \\ \text{subject to} & Ax = b \end{array}$$

is unique and given by

$$\hat{x} = A^T(AA^T)^{-1}b. \tag{10.1}$$

(Recall from section 3.5 that AA^T is positive definite if A is right-invertible.)

This result means that we can solve the least-norm problem by solving

$$(AA^T)z = b, \tag{10.2}$$

and then calculating $\hat{x} = A^T z$. The equations (10.2) are a set of m linear equations in m variables, and are called the *normal equations* associated with the least-norm problem.

Proof First, we check that $\hat{x}$ satisfies $A\hat{x} = b$. This is true because $A^T(AA^T)^{-1}$ is a right inverse of A:

$$A\hat{x} = (AA^T)(AA^T)^{-1}b = b.$$

Next, we show that any other solution of the equations has a norm greater than $\|\hat{x}\|$. Suppose x satisfies $Ax = b$. We have

$$\|x\|^2 = \|\hat{x} + (x - \hat{x})\|^2 = \|\hat{x}\|^2 + \|x - \hat{x}\|^2 + 2\hat{x}^T(x - \hat{x}).$$

The third term turns out to be zero, since

$$\begin{aligned}\hat{x}^T(x-\hat{x}) &= (A^T(AA^T)^{-1}b)^T(x-\hat{x}) \\ &= b^T(AA^T)^{-1}A(x-\hat{x}) \\ &= 0\end{aligned}$$

because $Ax = A\hat{x} = b$. We therefore have

$$\|x\|^2 = \|x-\hat{x}\|^2 + \|\hat{x}\|^2 \geq \|\hat{x}\|^2$$

with equality only if $x = \hat{x}$. In conclusion, if $Ax = b$ and $x \neq \hat{x}$ then

$$\|x\|^2 > \|\hat{x}\|^2.$$

This proves that $\hat{x}$ is the unique solution of the least-norm problem.

10.3 Solving least-norm problems by Cholesky factorization

We can solve the normal equations (10.2) using the Cholesky factorization of AA^T or the QR factorization of A^T.

Algorithm 10.1. Solving least-norm problems by Cholesky factorization.

given a right-invertible $m \times n$-matrix A and an m-vector b.

1. Form $C = AA^T$.
2. Compute the Cholesky factorization $C = LL^T$.
3. Solve $Lw = b$ by forward substitution.
4. Solve $L^Tz = w$ by back substitution.
5. Compute $x = A^Tz$.

The cost is nm^2 for step 1 (if we exploit the fact that C is symmetric), $(1/3)m^3$ for step 2, m^2 each for step 3 and step 4, and $2mn$ in step 5, which gives a total flop count of $nm^2 + (1/3)m^3 + 2m^2 + 2mn$, or roughly

$$nm^2 + (1/3)m^3.$$

Example We solve the least-norm problem defined by

$$A = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 0 & 1/2 & 1/2 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{10.3}$$

The normal equations $AA^Tz = b$ are

$$\begin{bmatrix} 4 & 2 \\ 2 & 3/2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The Cholesky factorization of AA^T is

$$\begin{bmatrix} 4 & 2 \\ 2 & 3/2 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 0 & 1/\sqrt{2} \end{bmatrix}.$$

To find z we first solve

$$\begin{bmatrix} 2 & 0 \\ 1 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

by forward substitution, which yields $w_1 = 0$, $w_2 = \sqrt{2}$. Next we solve

$$\begin{bmatrix} 2 & 1 \\ 0 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}$$

by back substitution and find $z_1 = -1$, $z_2 = 2$. Finally, we obtain the solution x from the matrix-vector product

$$x = A^T z = \begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 1 & 1/2 \\ 1 & 1/2 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

10.4 Solving least-norm problems by QR factorization

An alternative method is based on the QR factorization of the matrix A^T.

Suppose $A^T = QR$ with $Q^TQ = I$ and R upper triangular with positive diagonal elements. We can simplify the expression (10.1) as follows:

$$\begin{aligned} \hat{x} = A^T(AA^T)^{-1}b &= QR(R^TQ^TQR)^{-1}b \\ &= QR(R^TR)^{-1}b \\ &= QRR^{-1}R^{-T}b \\ &= QR^{-T}b. \end{aligned}$$

To find $\hat{x}$ we first compute $R^{-T}b$ (*i.e.*, solve the equation $R^Tz = b$), and then multiply with Q.

Algorithm 10.2. Solving least-norm problems by QR factorization.

given a right-invertible $m \times n$-matrix A and an m-vector b.

1. Compute the QR factorization $A^T = QR$.
2. Solve $R^Tz = b$ by forward substitution.
3. Compute $x = Qz$.

The cost is $2nm^2$ (for the QR factorization), plus m^2 for the forward substitution, plus $2mn$ for the matrix-vector product Qz. The total is $2nm^2 + m^2 + 2mn$ or roughly

$$2nm^2.$$

The advantages and disadvantages of the two methods are exactly the same as for least-squares problems. The QR factorization method is slower than the Cholesky factorization method (by a factor of about two if $n \gg m$), but it is more accurate. It is the preferred method if n and m are not too large. For very large sparse problems, the Cholesky factorization method is useful, because it is much more efficient than the QR factorization method.

Example We use the matrix A and b given in (10.3). The QR factorization of A^T is

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

The second step is to solve $R^T z = b$:

$$\begin{bmatrix} 2 & 0 \\ 1 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The solution is $z_1 = 0$, $z_2 = \sqrt{2}$. From z, we find the solution of the least-norm problem by evaluating $x = Qz$:

$$x = \begin{bmatrix} 1/2 & 1/\sqrt{2} \\ -1/2 & 1/\sqrt{2} \\ 1/2 & 0 \\ 1/2 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

10.5 Example

Solving least-norm problems in MATLAB If A is right-invertible, then the MATLAB command `x = A\b` computes a solution to $Ax = b$, but it is *not* the least-norm solution.

We can use the command `x = A'*((A*A')\b)` to compute the least-norm solution using the Cholesky factorization method. (MATLAB will recognize that AA^T is positive definite and use the Cholesky factorization to solve $AA^T z = b$.)

We can also use the QR factorization method, using the code

```
[Q,R] = qr(A');
Q = Q(:,1:m);
R = R(1:m:,:);
x = Q*(R'\b);
```

The second and third lines are added for the following reason. The definition of the QR factorization in MATLAB is slightly different from the definition in section 9. For a right-invertible $m \times n$ matrix A, the MATLAB command `[Q,R] = qr(A')` will return an $n \times n$ matrix Q and $n \times m$ matrix R that satisfy $A^T = QR$. The matrix R has the form

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

where R_1 is an $m \times m$ upper triangular matrix with nonzero diagonal elements. The matrix Q is orthogonal, so if we partition Q as

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix},$$

with $Q_1 \in \mathbf{R}^{n\times m}$, and $Q_2 \in \mathbf{R}^{n\times(n-m)}$, then

$$Q^TQ = \begin{bmatrix} Q_1^TQ_1 & Q_1^TQ_2 \\ Q_2^TQ_1 & Q_2^TQ_2 \end{bmatrix} = I.$$

It follows that Q_1 is orthogonal ($Q_1^TQ_1 = I$), and that

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1R_1.$$

We conclude that $A^T = Q_1R_1$ is the QR factorization according to our definition (except that we choose the triangular matrix to have positive diagonal elements, while MATLAB simply makes them nonzero, but that does not make much difference).

Example We consider a simple optimal control problem based on exercise 1.3. Suppose we are interested in finding an efficient (minimum-energy) force $F(t)$ that moves a unit mass in 10 steps to the position $s(10) = 1$, with final velocity $s'(10) = 0$. We assume that the total energy used is proportional to

$$\int_{t=0}^{10} F(t)^2\, dt = \sum_{i=1}^{10} x_i^2 = \|x\|^2.$$

In exercise 1.3 the following expressions for the velocity and position at time $t = 10$ were derived:

$$\begin{aligned} s'(10) &= x_1 + x_2 + \cdots + x_{10} \\ s(10) &= (19/2)x_1 + (17/2)x_2 + \cdots + (1/2)x_{10}. \end{aligned}$$

The control problem can therefore be posed as a least-norm problem

$$\begin{array}{ll} \text{minimize} & x_1^2 + x_2^2 + \cdots + x_9^2 + x_{10}^2 \\ \text{subject to} & \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 19/2 & 17/2 & \cdots & 3/2 & 1/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \end{array}$$

and solved in MATLAB as follows:

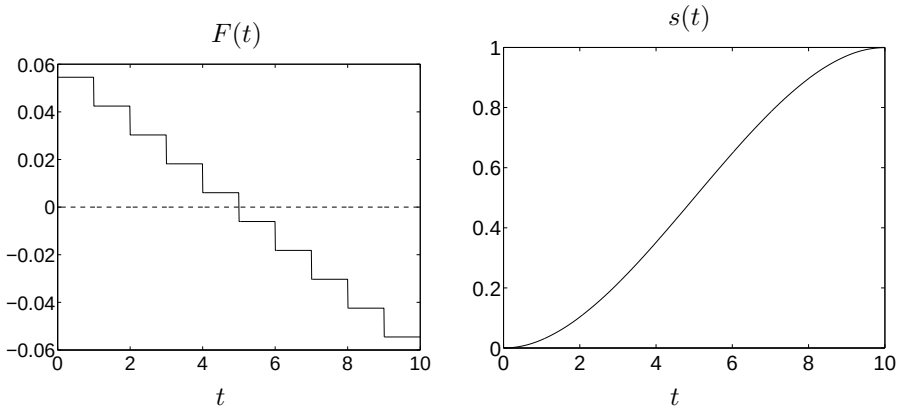


Figure 10.1 Left: The optimal force $F(t)$ that transfers the mass over a unit distance in 10 steps. Right: the resulting position of the mass $s(t)$.

```
A = [ones(1,10); (19/2):-1:(1/2)];
x = A'*((A*A')\[0;1]);
```

The solution x, and the resulting position $s(t)$, are shown in figure 10.1. The norm of the least-norm solution x is 0.1101.

It is interesting to compare the least norm solution of $Ax = b$ with a few other solutions. An obvious choice of x that also satisfies $Ax = b$ is $x = (1, -1, 0, \ldots, 0)$. The resulting position is shown in figure 10.2. This solution has norm $\sqrt{2}$, but requires only two time steps, so we might call it the 'minimum-time' solution.

Another solution is $x = (0, \ldots, 0, 1, -1)$, which also has norm $\sqrt{2}$.

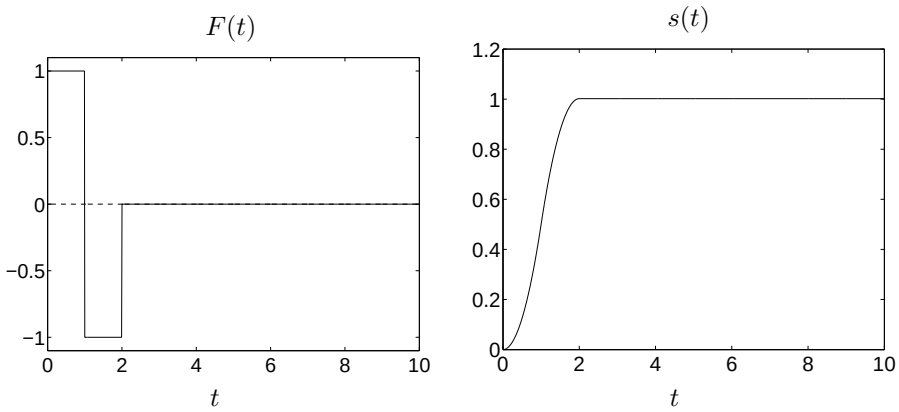


Figure 10.2 Left: A non-minimum norm force $F(t)$ that that transfers the mass over a unit distance in 10 steps. Right: the resulting position of the mass $s(t)$.

Exercises

Examples and applications

10.1 u, v, and w are three points in $\mathbf{R}^4$, given by

$$u = \begin{bmatrix} -1 \\ 7 \\ 1 \\ 1 \end{bmatrix}, \qquad v = \begin{bmatrix} -5 \\ -5 \\ 5 \\ 0 \end{bmatrix}, \qquad w = \begin{bmatrix} -2 \\ 9 \\ -1 \\ 0 \end{bmatrix}.$$

Each of the following four sets is a (two-dimensional) plane in $\mathbf{R}^4$.

(a) $\mathcal{H}_1 = \{x \in \mathbf{R}^4 \mid x = \alpha u + \beta v \text{ for some } \alpha, \beta \in \mathbf{R}\}$.

(b) $\mathcal{H}_2 = \{x \in \mathbf{R}^4 \mid x = \alpha u + \beta v + w \text{ for some } \alpha, \beta \in \mathbf{R}\}$.

(c) $\mathcal{H}_3 = \{x \in \mathbf{R}^4 \mid u^T x = 0 \text{ and } v^T x = 0\}$.

(d) $\mathcal{H}_4 = \{x \in \mathbf{R}^4 \mid u^T x = 1 \text{ and } v^T x = 1\}$.

For each set $\mathcal{H}_i$, find the projection of the point $y = (1, 1, 1, 1)$ on $\mathcal{H}_i$ (*i.e.*, find the point in $\mathcal{H}_i$ closest to y).

Hint. Formulate each problem as a least-squares or a least-norm problem, and solve it using MATLAB.

10.2 *Minimum-energy optimal control.* A simple model of a vehicle moving in one dimension is given by

$$\begin{bmatrix} s_1(t+1) \\ s_2(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0.95 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} u(t), \qquad t = 0, 1, 2, \ldots.$$

$s_1(t)$ is the position at time t, $s_2(t)$ is the velocity at time t, and $u(t)$ is the actuator input. Roughly speaking, the equations state that the actuator input affects the velocity, which in turn affects the position. The coefficient 0.95 means that the velocity decays by 5% in one sample period (for example, because of friction), if no actuator signal is applied. We assume that the vehicle is initially at rest at position 0: $s_1(0) = s_2(0) = 0$.

We will solve the *minimum energy optimal control problem*: for a given time horizon N, choose inputs $u(0), \ldots, u(N-1)$ so as to minimize the total energy consumed, which we assume is given by

$$E = \sum_{t=0}^{N-1} u(t)^2.$$

In addition, the input sequence must satisfy the constraint $s_1(N) = 10$, $s_2(N) = 0$. In other words, our task is to bring the vehicle to the final position $s_1(N) = 10$ with final velocity $s_2(N) = 0$, as efficiently as possible.

(a) Formulate the minimum energy optimal control problem as a least-norm problem

$$\begin{array}{ll} \text{minimize} & \|x\|^2 \\ \text{subject to} & Ax = b. \end{array}$$

Clearly state what the variables x, and the problem data A and b are.

(b) Solve the problem for $N = 30$. Plot the optimal $u(t)$, the resulting position $s_1(t)$, and velocity $s_2(t)$.

(c) Solve the problem for $N = 2, 3, \ldots, 29$. For each N calculate the energy E consumed by the optimal input sequence. Plot E versus N. (The plot will look better if you use a logarithmic scale for E, *i.e.*, `semilogy` instead of `plot`.)

(d) Suppose we allow the final position to deviate from 10. However, if $s_1(N) \neq 10$, we have to pay a penalty, equal to $(s_1(N) - 10)^2$. The problem is to find the input sequence that minimizes the sum of the energy E consumed by the input and the terminal position penalty,

$$\sum_{t=0}^{N-1} u(t)^2 + (s_1(N) - 10)^2,$$

subject to the constraint $s_2(N) = 0$.

Formulate this problem as a least-norm problem, and solve it for $N = 30$. Plot the optimal input signals $u(t)$, the resulting position $s_1(t)$ and the resulting velocity $s_2(t)$.

10.3 Two vehicles are moving along a straight line. For the first vehicle we use the same model as in exercise 10.2:

$$\begin{bmatrix} s_1(t+1) \\ s_2(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0.95 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} u(t), \qquad t = 0, 1, 2, \ldots,$$

$s_1(t)$ is the position at time t, $s_2(t)$ is the velocity at time t, and $u(t)$ is the actuator input. We assume that the vehicle is initially at rest at position 0: $s_1(0) = s_2(0) = 0$.

The model for the second vehicle is

$$\begin{bmatrix} p_1(t+1) \\ p_2(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0.8 \end{bmatrix} \begin{bmatrix} p_1(t) \\ p_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0.2 \end{bmatrix} v(t), \qquad t = 0, 1, 2, \ldots,$$

$p_1(t)$ is the position at time t, $p_2(t)$ is the velocity at time t, and $v(t)$ is the actuator input. We assume that the second vehicle is initially at rest at position 1: $p_1(0) = 1$, $p_2(0) = 0$.

Formulate the following problem as a least-norm problem, and solve it in MATLAB. Find the control inputs $u(0)$, $u(1)$, $\ldots$, $u(19)$ and $v(0)$, $v(1)$, $\ldots$, $v(19)$ that minimize the total energy

$$\sum_{t=0}^{19} u(t)^2 + \sum_{t=0}^{19} v(t)^2$$

and satisfy the following three conditions:

$$s_1(20) = p_1(20), \qquad s_2(20) = 0, \qquad p_2(20) = 0. \tag{10.4}$$

In other words, at time $t = 20$ the two vehicles must have velocity zero, and be at the same position. (The final position itself is not specified, *i.e.*, you are free to choose any value as long as $s_1(20) = p_1(20)$.)

Plot the positions $s_1(t)$ and $p_1(t)$ of the two vehicles, for $t = 1, 2, \ldots, 20$.

Solving least-norm problems via QR factorization

10.4 Explain how you would solve the following problems using the QR factorization. State clearly

- what the matrices are that you factor, and why you know that they have a QR factorization
- how you obtain the solution of each problem from the results of the QR factorizations
- what the cost (number of flops) is of your method.

(a) Find the solution of $Ax = b$ with the smallest value of $\sum_{i=1}^n d_i x_i^2$:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n d_i x_i^2 \\ \text{subject to} & Ax = b. \end{array}$$

The problem data are the $m \times n$-matrix A, the m-vector b, and the n-vector d. We assume that A is right-invertible, and $d_i > 0$ for all i.

(b) Find the solution of $Ax = b$ with the smallest value of $\|x - x_0\|^2$:

$$\begin{array}{ll} \text{minimize} & \|x - x_0\|^2 \\ \text{subject to} & Ax = b. \end{array}$$

The variable is the n-vector x. The problem data are the $m \times n$-matrix A, the m-vector b, and the n-vector x_0. We assume that A is right-invertible.

(c) Find the solution of $Ax = b$ with the smallest value of $\|Cx\|^2$:

$$\begin{array}{ll} \text{minimize} & \|Cx\|^2 \\ \text{subject to} & Ax = b. \end{array}$$

The problem data are the $p \times n$-matrix C, the $m \times n$-matrix A, and the m-vector b. We assume that A is right-invertible, and C is left-invertible.

(d) Find the solution of $Ax = b$ with the smallest value of $\|x\|^2 - c^T x$:

$$\begin{array}{ll} \text{minimize} & \|x\|^2 - c^T x \\ \text{subject to} & Ax = b. \end{array}$$

The problem data are the n-vector c, the $m \times n$-matrix A, and the m-vector b. We assume that A is right-invertible.

(e) Find the solution of $Ax = b$ with the smallest value of $\|C(x - x_0)\|^2$:

$$\begin{array}{ll} \text{minimize} & \|C(x - x_0)\|^2 \\ \text{subject to} & Ax = b. \end{array}$$

The $p \times n$-matrix C, the $m \times n$-matrix A, the n-vector x_0, and the m-vector b are given. We assume that A is right-invertible and C is left-invertible. Note that C is not necessarily square ($p \geq n$).

(f) Find the solution of $Ax = b$ with the smallest value of $\|Cx - d\|^2$:

$$\begin{array}{ll} \text{minimize} & \|Cx - d\|^2 \\ \text{subject to} & Ax = b. \end{array}$$

Th4 $p \times n$-matrix C, the $m \times n$-matrix A, the p-vector d and the m-vector b are given. We assume that A is right-invertible and C is left-invertible. Note that C is not necessarily square ($p \geq n$).

10.5 Show how to solve the following problems using the QR factorization of A. In each problem A is a left-invertible $m \times n$ matrix. Clearly state the different steps in your method. Also give a flop count, including all the terms that are quadratic (order m^2, mn, or n^2), or cubic (order m^3, m^2n, mn^2, n^3).

If you know several methods, give the most efficient one.

(a) Solve the set of linear equations

$$\begin{bmatrix} 0 & A^T \\ A & I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}.$$

The variables are the n-vector x and the m-vector y.

(b) Solve the least-squares problem

$$\text{minimize} \quad 2\|Ax-b\|^2 + 3\|Ax-c\|^2.$$

The variable is the n-vector x.

(c) Solve the least-norm problem

$$\begin{array}{ll} \text{minimize} & \|x\|^2 + \|y\|^2 \\ \text{subject to} & A^T x - 2A^T y = b. \end{array}$$

The variables are the m-vectors x and y.

(d) Solve the quadratic minimization problem

$$\text{minimize} \quad x^T A^T A x + b^T x + c.$$

The variable is the n-vector x.

10.6 If A is a left-invertible $m \times n$-matrix, and D is an $m \times m$ diagonal matrix with positive diagonal elements, then the coefficient matrix of the equation

$$\begin{bmatrix} D^2 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

is nonsingular. Therefore the equation has a unique solution $\hat{x}$, $\hat{y}$.

(a) Show that $\hat{x}$ is the solution of the optimization problem

$$\begin{array}{ll} \text{minimize} & \|Dx - D^{-1}b\|^2 \\ \text{subject to} & A^T x = c. \end{array}$$

(b) Show that $\hat{y}$ is the solution of the optimization problem

$$\text{minimize} \quad \|D^{-1}(Ay-b)\|^2 + 2c^T y.$$

(Hint: set the gradient of the cost function to zero.)

(c) Describe an efficient method, based on the QR factorization of $D^{-1}A$, for computing $\hat{x}$ and $\hat{y}$. Clearly state the different steps in your algorithm, the complexity of each step (number of flops for large m, n), and the total complexity.

10.7 Let A be an $m \times n$ matrix, b an n-vector, and suppose the QR factorization

$$\begin{bmatrix} A^T & b \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

exists. The matrix Q_1 has size $n \times m$, Q_2 has size $n \times 1$, R_{11} has size $m \times m$, R_{12} is $m \times 1$, and R_{22} is a scalar. Show that $\hat{x} = Q_2 R_{22}$ solves the optimization problem

$$\begin{array}{ll} \text{minimize} & \|x-b\|^2 \\ \text{subject to} & Ax = 0. \end{array}$$

10.8 Suppose A is a left-invertible matrix of size $m \times n$. Let $\hat{x}$ be the solution of the optimization problem

$$\text{minimize} \quad \|Ax-b\|^2 + 2c^T x$$

with $b \in \mathbf{R}^m$ and $c \in \mathbf{R}^n$, and let $\hat{y}$ be the solution of

$$\begin{array}{ll} \text{minimize} & \|y-b\|^2 \\ \text{subject to} & A^T y = c. \end{array}$$

(a) Show that $\hat{y} = b - A\hat{x}$. (*Hint.* To find an expression for $\hat{x}$, set the gradient of $\|Ax - b\|^2 + 2c^T x$ equal to zero.)

(b) Describe an efficient method for calculating $\hat{x}$ and $\hat{y}$ using the QR factorization of A. Clearly state the different steps in your algorithm and give a flop count, including all terms that are quadratic (m^2, mn, n^2) or cubic (m^3, m^2n, mn^2, n^3) in m and n.

10.9 Consider the underdetermined set of linear equations

$$Ax + By = b$$

where the p-vector b, the $p \times p$-matrix A, and the $p \times q$-matrix B are given. The variables are the p-vector x and the q-vector y. We assume that A is nonsingular, and that B is left-invertible (which implies $q \leq p$). The equations are underdetermined, so there are infinitely many solutions. For example, we can pick any y, and solve the set of linear equations $Ax = b - By$ to find x.

Below we define four solutions that minimize some measure of the magnitude of x, or y, or both. For each of these solutions, describe the factorizations (QR, Cholesky, or LU) that you would use to calculate x and y. Clearly specify the matrices that you factor, and the type of factorization. If you know several methods, you should give the most efficient one.

(a) The solution x, y with the smallest value of $\|x\|^2 + \|y\|^2$

(b) The solution x, y with the smallest value of $\|x\|^2 + 2\|y\|^2$.

(c) The solution x, y with the smallest value of $\|y\|^2$.

(d) The solution x, y with the smallest value of $\|x\|^2$.

Part III

Nonlinear equations and optimization

Chapter 11

Complexity of iterative algorithms

11.1 Iterative algorithms

The algorithms we discussed so far are non-iterative. They require a finite number of floating-point operations, that can be counted and expressed as a polynomial function of the problem dimensions. In chapters 12 to 14 we discuss problems that are solved by *iterative* algorithms. By this is meant an algorithm that computes a sequence of values $x^{(0)},\ x^{(1)}, x^{(2)}, \ldots$, with

$$x^{(k)} \to x^*$$

as $k \to \infty$, where the scalar or vector x^* is a solution of the problem. $x^{(0)}$ is called the *starting point* of the algorithm, and $x^{(k)}$ is called the kth *iterate*. Moving from $x^{(k)}$ to $x^{(k+1)}$ is called an *iteration* of the algorithm. The algorithm is terminated when $\|x^{(k)} - x^*\| \leq \epsilon$, where $\epsilon > 0$ is some specified tolerance, or when it is determined that the sequence is not converging (for example, when a limit on the number of iterations is exceeded).

The total cost of an iterative algorithm is more difficult to estimate than the cost of a non-iterative algorithm, because the number of iterations depends on the problem parameters and on the starting point. The efficiency of an iterative algorithm is therefore usually not expressed by giving its flop count, but by giving upper bounds on the number of iterations to reach a given accuracy. Deriving such bounds is the purpose of *convergence analysis*.

Iterative algorithms are often classified according to their rate of convergence. In the following paragraphs we give an overview of the most common definitions. For simplicity we will assume x^* is a scalar, and $x^{(k)}$ $(k = 0, 1, 2, \ldots)$ is a sequence of numbers converging to $x^\star$.

Absolute and relative error The error after k iterations can be expressed as the *absolute error*, $|x^{(k)} - x^\star|$, or as the *relative error* $|x^{(k)} - x^\star|/|x^\star|$ (which is defined only if $x^\star \neq 0$). The relative error can also be expressed as the *number of correct*

digits, for which several slightly different definitions exist. The definition that we will adopt is the following: if the relative error $|x^{(k)} - x^\star|/|x^\star|$ is less than one, then the number of correct digits in $x^{(k)}$ is defined as

$$\left\lfloor -\log_{10}\left(\frac{|x^{(k)} - x^\star|}{|x^\star|}\right)\right\rfloor$$

(by $\lfloor\alpha\rfloor$ we mean the largest integer less than or equal to α). In other words, we take the logarithm with base 10 of the relative error (which is a negative number if the relative error is less than one), change its sign, and round it down to the nearest integer. This means that r correct digits correspond to relative errors in the interval $[10^{-r}, 10^{-r-1})$.

As an example, the table shows three numbers x close to $x^\star = \pi = 3.141592\ldots$, the relative error $|x - \pi|/\pi$, and the number of correct digits $\lfloor -\log_{10}(|x - \pi|/\pi)\rfloor$.

$x^{(k)}$	rel. error	#correct digits
3.14150	$2.9 \cdot 10^{-5}$	4
3.14160	$2.3 \cdot 10^{-6}$	5
3.14165	$1.8 \cdot 10^{-5}$	4

Note that other, perhaps more intuitive, definitions of correct digits might give slightly different results. For example, if we simply count the number of leading digits where π and x agree, we would say there are 5 correct digits in 3.14150, and 4 correct digits in 3.14160 and 3.14165. On the other hand, we might also argue that the number of correct digits in $x = 3.14150$ is actually 4, because if we round x and π to 4 digits, we obtain the same number (3.142), but if we round them to 5 digits, they are different (3.1415 and 3.1416).

These discrepancies do not matter in practice. If we use the number of correct digits to describe the accuracy of an approximation, it is understood that we are only giving a rough indication of the relative error. 'Four correct digits' means the relative error is roughly 10^{-4}. It could be a little more or little less, depending on the definition, but that should not matter, because if we want to say exactly what the accuracy is, we can simply give the relative (or absolute) error.

11.2 Linear and R-linear convergence

A sequence $x^{(k)}$ with limit $x^\star$ is *linearly convergent* if there exists a constant $c \in (0, 1)$ such that

$$|x^{(k)} - x^\star| \leq c\,|x^{(k-1)} - x^\star| \tag{11.1}$$

for k sufficiently large. For example, the sequence $x^{(k)} = 1 + (1/2)^k$ converges linearly to $x^\star = 1$, because

$$|x^{(k+1)} - x^\star| = (1/2)^{k+1} = \frac{1}{2}|x^{(k)} - x^\star|$$

so the definition is satisfied with $c = 1/2$.

k	$1 + 0.5^k$
0	2.00000000000000
1	1.50000000000000
2	1.25000000000000
3	1.12500000000000
4	1.06250000000000
5	1.03125000000000
6	1.01562500000000
7	1.00781250000000
8	1.00390625000000
9	1.00195313125000
10	1.00097656250000

Table 11.1 The first ten values of $x^{(k)} = 1 + 1/2^k$.

If $x^\star \neq 0$, we can give an intuitive interpretation of linear convergence in terms of the number of correct digits in $x^{(k)}$. Let

$$r^{(k)} = -\log_{10} \frac{|x^{(k)} - x^\star|}{|x^\star|}.$$

Except for rounding to an integer, $r^{(k)}$ is the number of correct digits in $x^{(k)}$. If we divide both sides of the inequality (11.1) by $|x^\star|$ and take logarithms, we obtain

$$r^{(k+1)} \geq r^{(k)} - \log_{10} c.$$

Ignoring the effect of rounding, we can say we gain at least $-\log_{10} c$ correct digits per iteration.

We can verify this using the example $x^{(k)} = 1 + 1/2^k$. As we have seen, this sequence is linearly convergent with $c = 1/2$, so we expect to gain roughly $-\log_{10} 1/2 = 0.3$ correct digits per iteration, or in other words, one correct digit per three or four iterations. This is confirmed by table 11.1, which shows the first ten values of $x^{(k)}$.

R-linear convergence Linear convergence is also sometimes defined as follows. A sequence $x^{(k)}$ with limit $x^\star$ is *R-linearly convergent* if there exists a positive M and $c \in (0, 1)$ such that

$$|x^{(k)} - x^\star| \leq Mc^k \tag{11.2}$$

for sufficiently large k. This means that for large k the error decreases at least as fast as the geometric series Mc^k. We refer to this as R-linear convergence to distinguish it from the first definition. Every linearly convergent sequence is also R-linearly convergent, but the converse is not true. For example, the error in an R-linearly convergent sequence does not necessarily decrease monotonically, while the inequality (11.1) implies that $|x^{(k)} - x^\star| < |x^{(k-1)} - x^\star|$ for sufficiently large k.

k	$1 + 0.5^{2^k}$
0	1.50000000000000
1	1.25000000000000
2	1.06250000000000
3	1.00390625000000
4	1.00001525878906
5	1.00000000023283
6	1.00000000000000

Table 11.2 The first six values of $x^{(k)} = 1 + (1/2)^{2^k}$.

11.3 Quadratic convergence

A sequence $x^{(k)}$ with limit $x^\star$ is *quadratically convergent* if there exists a constant $c > 0$ such that

$$|x^{(k)} - x^\star| \leq c\, |x^{(k-1)} - x^\star|^2 \tag{11.3}$$

for k sufficiently large. The sequence $x^{(k)} = 1 + (1/2)^{2^k}$ converges quadratically to $x^\star = 1$, because

$$|x^{(k+1)} - x^\star| = (1/2)^{2^{k+1}} = \left((1/2)^{2^k}\right)^2 = |x^{(k)} - x^\star|^2,$$

so the definition is satisfied with $c = 1$.

If $x^\star \neq 0$, we can relate the definition to the number of correct digits in $x^{(k)}$. If we define $r^{(k)}$ as above, we can write the inequality (11.3) as

$$r^{(k)} \geq 2r^{(k-1)} - \log_{10}(|x^\star c|).$$

Since $|x^{(k)} - x^\star| \to 0$, we must have $r^{(k)} \to +\infty$, so sooner or later the first term on the right-hand side will dominate the second term, which is constant. For sufficiently large k, the number of correct digits roughly doubles in each iteration.

Table 11.2 shows the first few values of the sequence $x^{(k)} = 1 + (1/2)^{2^k}$ which converges quadratically with $c = 1$, and $x^\star = 1$. We start with one correct digit. It takes two iterations to get the second correct digit. The next iteration we gain one digit, then we gain two in one iteration, etc.

11.4 Superlinear convergence

A sequence $x^{(k)}$ with limit $x^\star$ is *superlinearly convergent* if there exists a sequence $c_k > 0$ with $c_k \to 0$ such that

$$|x^{(k)} - x^\star| \leq c_k\, |x^{(k-1)} - x^\star| \tag{11.4}$$

for sufficiently large k.

The sequence $x^{(k)} = 1 + (1/(k+1))^k$ is superlinearly convergent because

$$|x^{(k)} - x^\star| = \frac{1}{(k+1)^k} = \frac{k^{k-1}}{(k+1)^k}\frac{1}{k^{k-1}} = \frac{k^{k-1}}{(k+1)^k}|x^{(k-1)} - x^\star|,$$

so the definition is satisfied with $c_k = k^{k-1}/(k+1)^k$, which indeed goes to zero.

If we define $r^{(k)}$ as above, we can write the inequality (11.3) as

$$r^{(k)} \geq r^{(k-1)} - \log_{10}(c_k),$$

and since $c_k \to 0$, $-\log_{10} c_k \to \infty$. For sufficiently large k, the number of correct digits we gain per iteration ($-\log_{10}(c_k)$) increases with k.

Table 11.3 shows the first values of $1 + (1/(k+1))^k$. The number of correct digits increases faster than linearly, but does not quite double per iteration.

k	$1 + (1/(k+1)^k)$
0	2.00000000000000
1	1.50000000000000
2	1.11111111111111
3	1.01562500000000
4	1.00160000000000
5	1.00012860082305
6	1.00000849985975
7	1.00000047683716
8	1.00000002323057
9	1.00000000100000
10	1.00000000003855

Table 11.3 The first ten values of $x^{(k)} = 1 + (1/(k+1))^k$.

Chapter 12

Nonlinear equations

In this chapter we discuss methods for finding a solution of n nonlinear equations in n variables

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0. \end{aligned}$$

To simplify notation, we will often express this as

$$f(x) = 0$$

where

$$f(x) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

We assume that f is at least continuous (*i.e.*, $\lim_{y \to x} f(y) = f(x)$ for all x), and for some algorithms, stronger assumptions will be needed (for example, differentiability).

We will focus on simple algorithms that are easy to program (in less than ten lines of MATLAB code), but may require some 'baby-sitting', typically in the form of a good starting point.

12.1 Iterative algorithms

Nonlinear equations usually have to be solved by iterative algorithms, that generate a sequence of points $x^{(0)},\ x^{(1)}, x^{(2)}, \ldots$ with

$$f(x^{(k)}) \to 0$$

as $k \to \infty$. The vector $x^{(0)}$ is called the *starting point* of the algorithm, and $x^{(k)}$ is called the kth *iterate*. Moving from $x^{(k)}$ to $x^{(k+1)}$ is called an *iteration* of the algorithm. The algorithm is terminated when $\|f(x^{(k)})\| \leq \epsilon$, where $\epsilon > 0$ is some specified tolerance, or when it is determined that the sequence is not converging (for example, when a limit on the number of iterations is exceeded).

Problem description In the simplest case, the functions f_i are given by a formula, which may depend on some parameters. Suppose, for example, that we are asked to write a program to solve n quadratic equations

$$\begin{aligned} f_1(x) &= x^T P_1 x + q_1^T x + r_1 \\ f_2(x) &= x^T P_2 x + q_2^T x + r_2 \\ &\vdots \\ f_n(x) &= x^T P_n x + q_n^T x + r_n. \end{aligned}$$

The parameters in the problem (or the problem data) are the coefficients of the matrices P_i, the coefficients of the vectors q_i, and the numbers r_i. To define a specific problem of this form, we can give the values of all the parameters.

In other cases the functions f_i are described by *oracle* models (also called *black box* or *subroutine* models). In an oracle model, we do not know f explicitly, but can evaluate $f(x)$ (and usually also some derivatives) at any x. This is referred to as *querying the oracle*, and is usually associated with some cost, *e.g.*, time. As a concrete example, suppose we are given a subroutine that evaluates f and its derivatives. We can call the subroutine at any x, but do not have access to its source code. In this model, we never really know the function; we only know the function value (and some derivatives) at the points at which we have queried the oracle. An algorithm is then judged, in part, on the number of times the oracle must be queried before an approximate solution (with $\|f(x)\| \leq \epsilon$) is found.

12.2 Bisection method

The first method we discuss only applies to problems with $n = 1$.

We start with an interval $[l, u]$ that satisfies $f(l)f(u) < 0$ (the function values at the end points of the interval have opposite signs). Since f is continuous, this guarantees that the interval contains at least one solution of $f(x) = 0$.

In each iteration we evaluate f at the midpoint $(l + u)/2$ of the interval, and depending on the sign of $f((l + u)/2)$, replace l or u with $(l + u)/2$. If $f((u + l)/2)$ has the same sign as $f(l)$, we replace l with $(u + l)/2$. Otherwise we replace u. Thus we obtain a new interval that still satisfies $f(l)f(u) < 0$. The method is called bisection because the interval is replaced by either its left or right half at each iteration.

Algorithm 12.1. BISECTION ALGORITHM.

given l, u with $l < u$ and $f(l)f(u) < 0$; a required tolerance $\epsilon > 0$
repeat

1. $x := (l + u)/2$.
2. Compute $f(x)$.
3. **if** $f(x) = 0$, **return** x.
4. **if** $f(x)f(l) < 0$, $u := x$, **else**, $l := x$.

until $u - l \leq \epsilon$

The convergence of the bisection method is easy to analyze. If we denote by $[l^{(0)}, u^{(0)}]$ the initial interval, and by $[l^{(k)}, u^{(k)}]$ the interval after iteration k, then

$$u^{(k)} - l^{(k)} = \frac{u^{(0)} - l^{(0)}}{2^k}, \tag{12.1}$$

because the length of the interval is divided by two at each iteration. This means that the exit condition $u^{(k)} - l^{(k)} \leq \epsilon$ will be satisfied if

$$\log_2(\frac{u^{(0)} - l^{(0)}}{2^k}) = \log_2(u^{(0)} - l^{(0)}) - k \leq \log_2 \epsilon,$$

i.e., as soon as $k \geq \log_2((u^{(0)} - l^{(0)})/\epsilon)$. The algorithm therefore terminates after

$$\left\lceil \log_2 \left(\frac{u^{(0)} - l^{(0)}}{\epsilon} \right) \right\rceil$$

iterations. (By $\lceil \alpha \rceil$ we mean the smallest integer greater than or equal to α.)

Since the final interval contains at least one solution $x^\star$, we are guaranteed that its midpoint

$$x^{(k)} = \frac{1}{2}(l^{(k)} + u^{(k)})$$

is no more than a distance $u^{(k)} - l^{(k)}$ from $x^\star$. Thus we have

$$|x^{(k)} - x^\star| \leq \epsilon,$$

when the algorithm terminates.

The advantages of the bisection method are its simplicity, and the fact that it does not require derivatives. It also does not require a starting point close to $x^\star$. The disadvantages are that it is not very fast, and that it does not extend to $n > 1$. Selecting an initial interval that satisfies $f(l)f(u) < 0$ may also be difficult.

Convergence rate The bisection method is *R-linearly convergent* (as defined in section 11.2). After k iterations, the midpoint $x^{(k)} = (u^{(k)} + l^{(k)})/2$ satisfies

$$|x^{(k)} - x^\star| \leq u^{(k)} - l^{(k)} \leq (1/2)^k(u^{(0)} - l^{(0)}),$$

(see equation (12.1)). Therefore the definition (11.2) is satisfied with $c = 1/2$ and $M = u^{(0)} - l^{(0)}$.

12.3 Newton's method for one equation with one variable

Newton's method is the most popular method for solving nonlinear equations. We first explain the method for $n = 1$, and then extend it to $n > 1$. We assume that f is differentiable.

Algorithm 12.2. NEWTON'S METHOD FOR ONE EQUATION WITH ONE VARIABLE.

given initial x, required tolerance $\epsilon > 0$
repeat

1. Compute $f(x)$ and $f'(x)$.
2. **if** $|f(x)| \leq \epsilon$, **return** x.
3. $x := x - f(x)/f'(x)$.

until maximum number of iterations is exceeded.

For simplicity we assume that $f'(x) \neq 0$ in step 3. (In a practical implementation we would have to make sure that the code handles the case $f'(x) = 0$ gracefully.)

The algorithm starts at some initial value $x^{(0)}$, and then computes iterates $x^{(k)}$ by repeating

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k = 0, 1, 2, \ldots.$$

This update has a simple interpretation. After evaluating the function value $f(x^{(k)})$ and the derivative $f'(x^{(k)})$, we construct the first-order Taylor approximation to f around $x^{(k)}$:

$$f_{\text{aff}}(y) = f(x^{(k)}) + f'(x^{(k)})(y - x^{(k)}).$$

We then solve $f_{\text{aff}}(y) = 0$, *i.e.*,

$$f(x^{(k)}) + f'(x^{(k)})(y - x^{(k)}) = 0,$$

for the variable y. This is called the *linearized equation*. If $f'(x^{(k)}) \neq 0$, the solution exists and is given by

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

We then take y as the next value $x^{(k+1)}$. This is illustrated in figure 12.1.

Examples Simple examples will illustrate the convergence properties of Newton's method. We first consider the nonlinear equation

$$f(x) = e^x - e^{-x} - 1 = 0.$$

The derivative is $f'(x) = e^x + e^{-x}$, so the Newton iteration is

$$x^{(k+1)} = x^{(k)} - \frac{e^{x^{(k)}} - e^{-x^{(k)}} - 1}{e^{x^{(k)}} + e^{-x^{(k)}}}, \quad k = 0, 1, \ldots$$

If we start at $x = 4$, the algorithm converges very quickly to $x^\star = 0.4812$ (see figure 12.2).

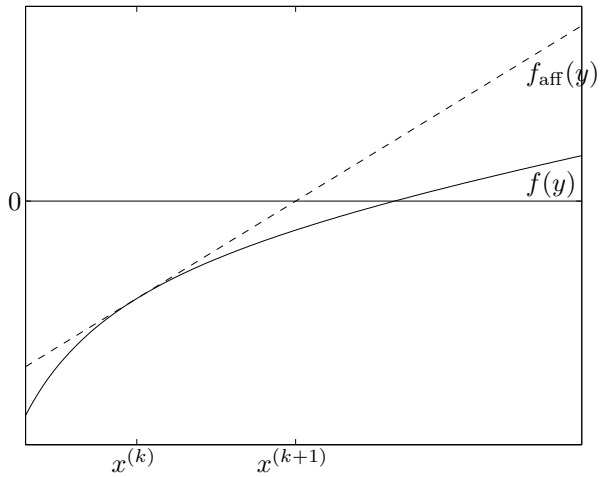


Figure 12.1 One iteration of Newton's method. The iterate $x^{(k+1)}$ is the zero-crossing of the first-order Taylor approximation of f at $x^{(k)}$.

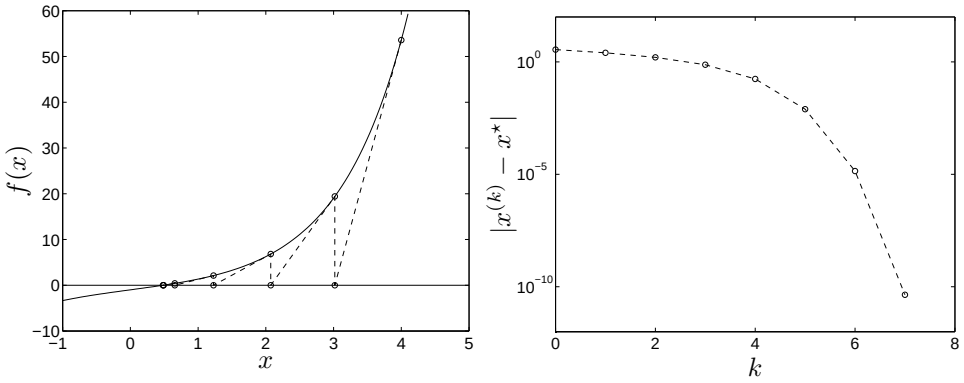


Figure 12.2 The solid line in the left plot is $f(x) = e^x - e^{-x} - 1$. The dashed line and the circles indicate the iterates in Newton's method for solving $f(x) = 0$, starting at $x^{(0)} = 4$. The right plot shows the error $|x^{(k)} - x^\star|$ versus k.

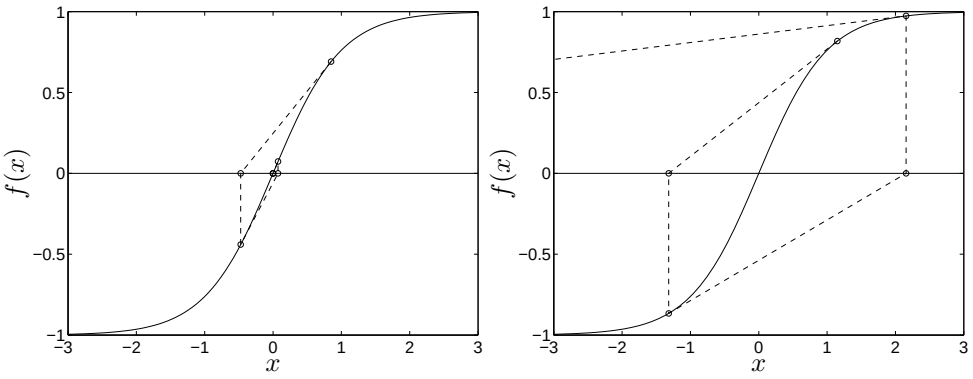


Figure 12.3 The solid line in the left plot is $f(x) = (e^x - e^{-x})/(e^x + e^{-x})$. The dashed line and the circles indicate the iterates in Newton's method for solving $f(x) = 0$, starting at $x^{(0)} = 0.85$ (left) and $x^{(0)} = 1.15$ (right). In the first case the method converges rapidly to $x^\star = 0$. In the second case it does not converge.

As a second example, we consider the equation

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 0.$$

The derivative is $f'(x) = 4/(e^x + e^{-x})^2$, so the Newton iteration is

$$x^{(k+1)} = x^{(k)} - \frac{1}{4}(e^{2x^{(k)}} - e^{-2x^{(k)}}), \quad k = 0, 1, \ldots$$

Figure 12.3 shows the iteration, starting at two starting points, $x^{(0)} = 0.85$, and $x^{(0)} = 1.15$. The method converges rapidly from $x^{(0)} = 0.85$, but does not converge from $x^{(0)} = 1.15$.

The two examples are typical for the convergence behavior of Newton's method: it works very well if started near a solution; it may not work at all when started far from a solution.

12.4 Newton's method for sets of nonlinear equations

We now extend Newton's method to a nonlinear equation $f(x) = 0$ where $f : \mathbf{R}^n \to \mathbf{R}^n$ (a function that maps an n-vector x to an n-vector $f(x)$).

We start with an initial point $x^{(0)}$. At iteration k we evaluate $f(x^{(k)})$, the derivative matrix $Df(x^{(k)})$, and form the first-order approximation of f at $x^{(k)}$:

$$f_{\text{aff}}(y) = f(x^{(k)}) + Df(x^{(k)})(y - x^{(k)}).$$

We set $f_{\text{aff}}(y) = 0$ and solve for y, which gives

$$y = x^{(k)} - Df(x^{(k)})^{-1} f(x^{(k)})$$

(assuming $Df(x^{(k)})$ is nonsingular). This value is taken as the next iterate $x^{(k+1)}$. In summary,

$$x^{(k+1)} = x^{(k)} - Df(x^{(k)})^{-1} f(x^{(k)}), \quad k = 0, 1, 2, \ldots$$

Algorithm 12.3. NEWTON'S METHOD FOR SETS OF NONLINEAR EQUATIONS.

given an initial x, a required tolerance $\epsilon > 0$
repeat

1. Evaluate $g = f(x)$ and $H = Df(x)$.
2. **if** $\|g\| \leq \epsilon$, **return** x.
3. Solve $Hv = -g$.
4. $x := x + v$.

until maximum number of iterations is exceeded.

Example As an example, we take a problem with two variables

$$f_1(x_1, x_2) = \log(x_1^2 + 2x_2^2 + 1) - 0.5, \qquad f_2(x_1, x_2) = -x_1^2 + x_2 + 0.2. \tag{12.2}$$

There are two solutions, $(0.70, 0.29)$ and $(-0.70, 0.29)$. The derivative matrix is

$$Df(x) = \begin{bmatrix} 2x_1/(x_1^2 + 2x_2^2 + 1) & 4x_2/(x_1^2 + 2x_2^2 + 1) \\ -2x_1 & 1 \end{bmatrix}.$$

Figure 12.4 shows what happens if we use three different starting points.

The example confirms the behavior we observed for problems with one variable. Newton's method does not always work, but if started near a solution, it takes only a few iterations. The main advantage of the method is its very fast local convergence. The disadvantages are that it requires a good starting point, and that it requires the n^2 partial derivatives of f.

Many techniques have been proposed to improve the convergence properties. (A simple idea for $n = 1$ is to combine it with the bisection method.) These *globally convergent* Newton methods are designed in such a way that locally, in the neighborhood of a solution, they automatically switch to the standard Newton method.

A variation on Newton's method that does not require derivatives is the secant method, discussed in the next paragraph.

12.5 Secant method

Although the idea of the secant method extends to problems with several variables, we will only describe it for $n = 1$.

The secant method can be interpreted as a variation of Newton's method in which we replace the first-order approximation

$$f_{\text{aff}}(y) = f(x^{(k)}) + f'(x^{(k)})(y - x^{(k)})$$

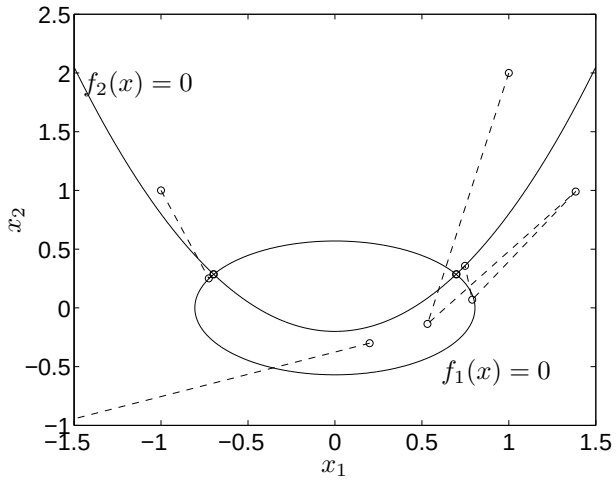


Figure 12.4 The solid lines are the zero-level curves of the functions f_1 and f_2 in (12.2). The circles are the iterates of Newton's method from three starting points. With $x^{(0)} = (-1, 1)$, the method converges to the solution $(-0.70, 0.29)$ in 2 or 3 steps. With $x^{(0)} = (1, 2)$, it converges to the solution $(0.70, 0.29)$ in about 5 iterations. With $x^{(0)} = (0.2, -0.3)$, it does not converge.

with the function

$$f_{\text{aff}}(y) = f(x^{(k)}) + \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}(y - x^{(k)}).$$

This is the affine function that agrees with $f(y)$ at $y = x^{(k)}$ and $y = x^{(k-1)}$. We then set $f_{\text{aff}}(y)$ equal to zero, solve for y, and take the solution as $x^{(k+1)}$ (figure 12.5).

Algorithm 12.4. SECANT METHOD FOR ONE EQUATION WITH ONE VARIABLE.

given two initial points x, x^-, required tolerance $\epsilon > 0$
repeat

1. Compute $f(x)$
2. **if** $|f(x)| \leq \epsilon$, **return** x.
3. $g := (f(x) - f(x^-))/(x - x^-)$.
4. $x^- := x$.
5. $x := x - f(x)/g$.

until maximum number of iterations is exceeded.

The convergence of the secant method is slower than the Newton method, but it does not require derivatives, so the amount of work per iteration is smaller.

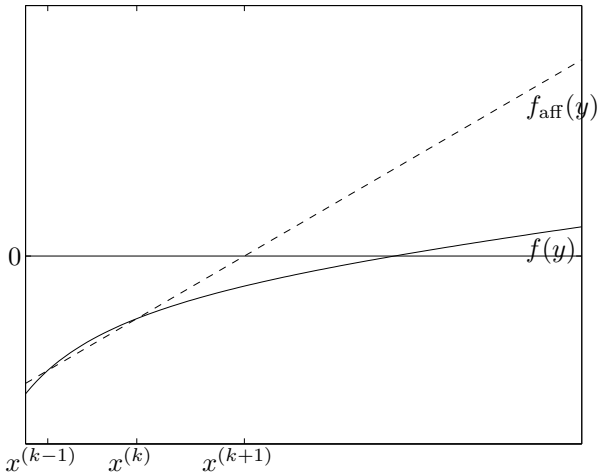


Figure 12.5 One iteration of the secant method. The iterate $x^{(k+1)}$ is the zero-crossing of the affine function through the points $(x^{(k-1)}, f(x^{(k-1)}))$ and $(x^{(k)}, f(x^{(k)}))$.

Example

Figure 12.6 shows the convergence of the secant method for the same example as in figure 12.2, with starting points $x^{(0)} = 4$, $x^{(-1)} = 4.5$.

12.6 Convergence analysis of Newton's method

Newton's method converges quadratically if $f'(x^\star) \neq 0$, and we start sufficiently close to $x^\star$. More precisely we can state the following result.

Suppose $I = [x^\star - \delta, x^\star + \delta]$ is an interval around a solution $x^\star$ on which f satisfies the following two properties.

1. There exists a constant $m > 0$ such that $|f'(x)| \geq m$ for all $x \in I$.

2. There exists a constant $L > 0$ such that $|f'(x) - f'(y) \leq L|x - y|$ for all $x, y \in I$.

We will show that if $|x^{(k)} - x^\star| \leq \delta$, then

$$|x^{(k+1)} - x^\star| \leq \frac{L}{2m}|x^{(k)} - x^\star|^2. \tag{12.3}$$

In other words, the inequality (11.3) is satisfied with $c = L/(2m)$, so if $x^{(k)}$ converges to $x^\star$, it converges quadratically.

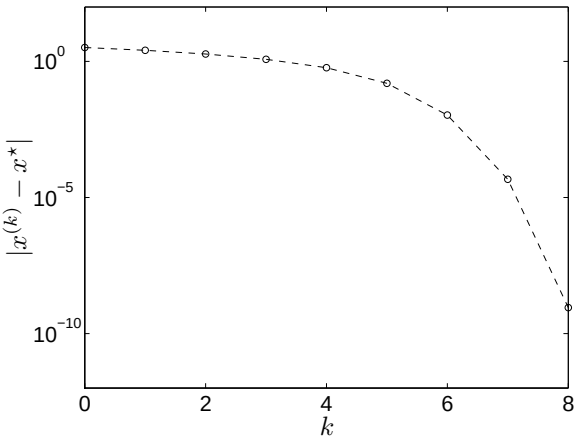


Figure 12.6 Error $|x^{(k)} - x^\star|$ versus k for the secant method applied to the equation $e^x - e^{-x} - 1 = 0$, with starting points $x^{(0)} = 4$, $x^{(-1)} = 4.5$.

The inequality (12.3) is proved as follows. For simplicity we will denote $x^{(k)}$ as x and $x^{(k+1)}$ as x^+, *i.e.*,

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

We have

$$\begin{aligned} |x^+ - x^\star| &= |x - \frac{f(x)}{f'(x)} - x^\star| \\ &= \frac{|-f(x) - f'(x)(x^\star - x)|}{|f'(x)|} \\ &= \frac{|f(x^\star) - f(x) - f'(x)(x^\star - x)|}{|f'(x)|}. \end{aligned}$$

(Recall that $f(x^\star) = 0$.) We have $|f'(x)| \geq m$ by the first property, and hence

$$|x^+ - x^\star| \leq \frac{|f(x^\star) - f(x) - f'(x)(x^\star - x)|}{m}. \tag{12.4}$$

We can use the second property to bound the numerator:

$$\begin{aligned} |f(x^\star) - f(x) - f'(x)(x^\star - x)| &= \left| \int_x^{x^\star} (f'(u) - f'(x))\, du \right| \\ &\leq \int_x^{x^\star} |f'(u) - f'(x)|\, du \\ &\leq L \int_x^{x^\star} |u - x|\, du \\ &= \frac{L}{2}|x^\star - x|^2. \end{aligned} \tag{12.5}$$

Putting (12.4) and (12.5) together, we obtain $|x^+ - x| \leq L|x^\star - x|^2/(2m)$, which proves (12.3).

The inequality (12.3) by itself does not guarantee that $|x^{(k)} - x^\star| \to 0$. However, if we assume that $|x^{(0)} - x^\star| \leq \delta$ and that $\delta \leq m/L$, then convergence readily follows. Since $|x^{(0)} - x^\star| \leq \delta$, we can apply the inequality (12.3) to the first iteration, which yields

$$|x^{(1)} - x^\star| \leq \frac{L}{2m}|x^{(0)} - x^\star|^2 \leq \frac{L}{2m}\delta^2 \leq \frac{\delta}{2}.$$

Therefore $|x^{(1)} - x^\star| \leq \delta$, so we can apply the inequality to $k = 1$, and obtain a bound on the error in $x^{(2)}$,

$$|x^{(2)} - x^\star| \leq \frac{L}{2m}|x^{(1)} - x^\star|^2 \leq \frac{L}{2m}\frac{\delta^2}{4} \leq \frac{\delta}{8},$$

and therefore also

$$|x^{(3)} - x^\star| \leq \frac{L}{2m}|x^{(2)} - x^\star|^2 \leq \frac{L}{2m}\frac{\delta^2}{8^2} \leq \frac{\delta}{128}$$

et cetera. Continuing in this fashion, we have

$$|x^{(k+1)} - x^\star| \leq \frac{L}{2m}|x^{(k)} - x^\star|^2 \leq 2\left(\frac{1}{4}\right)^{2^k}\delta,$$

which shows that the error converges to zero very rapidly.

A final note on the practical importance of this (and most other) convergence results. If $f'(x^\star) \neq 0$ and $f'(x)$ is a continuous function, then it is reasonable to assume that the assumptions we made are satisfied for *some* δ, m, and L. In practice, of course, we almost never know δ, m, L, so the convergence result does not provide any practical guidelines that might help us, for example, when selecting a starting point.

The result does provide some interesting qualitative or conceptual information. It establishes convergence of Newton's method, provided we start sufficiently close to a solution. It also explains the very fast local convergence observed in practice. As an interesting detail that we have not observed so far, the proof suggests that quadratic convergence only occurs if $f'(x^\star) \neq 0$. A simple example will confirm this.

Suppose we apply Newton's method to the nonlinear equation

$$f(x) = x^2 - a = 0,$$

where a is a nonnegative number. Newton's method uses the iteration

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \frac{(x^{(k)})^2 - a}{2x^{(k)}} \\ &= \frac{1}{2}(x^{(k)} + \frac{a}{x^{(k)}}), \quad k = 0, 1, 2, \ldots. \end{aligned}$$

Figures 12.7 and 12.8 show the result for $a = 5$ and $a = 0$ respectively, with starting point $x^{(0)} = 5$. Newton's method converges in both cases, but much more slowly when $a = 0$ (and hence $f'(x^\star) = 0$).

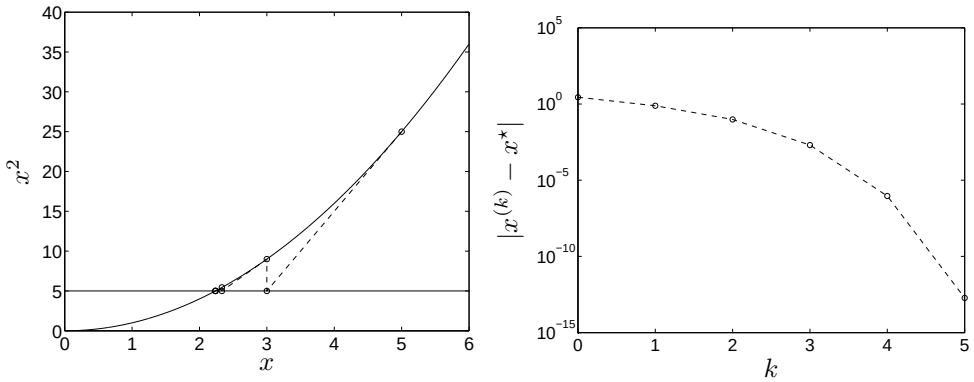


Figure 12.7 The solid line on the left is the function x^2. The circles and dashed lines show the iterates in Newton's method applied to the function $x^2 - 5 = 0$. The right plot shows the error versus k.

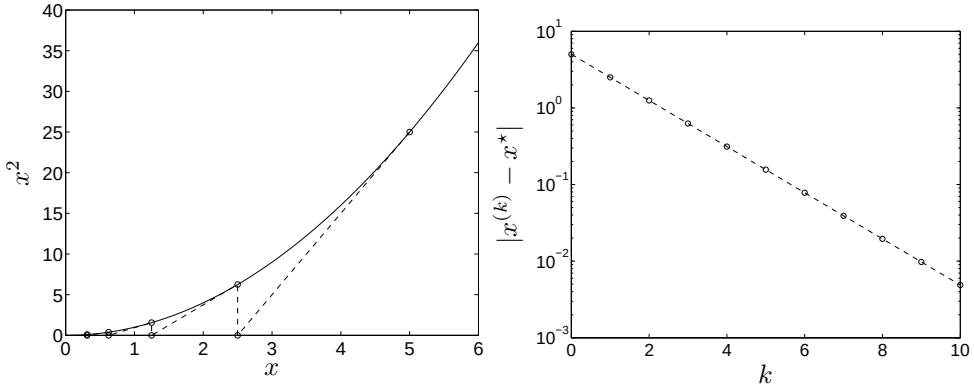


Figure 12.8 The solid line on the left is the function x^2. The circles and dashed lines show the iterates in Newton's method applied to the function $x^2 = 0$. The right plot shows the error versus k.

Sets of equations The quadratic convergence result for Newton's method generalizes to functions of several variables. The precise statement is as follows. Suppose there is a neighborhood

$$I = \{x \mid \|x - x^\star\| \leq \delta\}$$

around a solution $x^\star$ on which f satisfies the following two properties.

1. There exists a constant $m > 0$ such that $\|Df(x)^{-1}\| \leq 1/m$ for all $x \in I$.
2. There exists a constant $L > 0$ such that $\|Df(x) - Df(y)\| \leq L\|x - y\|$ for all $x, y \in I$.

(Note that the norms $\|Df(x)^{-1}\|$ and $\|Df(x) - Df(y)\|$ are matrix norms, and $\|x - y\|$ is a vector norm.) If $\|x^{(k)} - x^\star\| \leq \delta$, then

$$\|x^{(k+1)} - x^\star\| \leq \frac{L}{2m}\|x^{(k)} - x^\star\|^2.$$

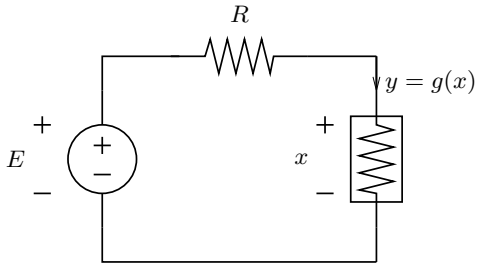
Secant method Under similar assumptions as Newton's method (including, in particular, $f'(x^\star) \neq 0$ and a starting point sufficiently close to $x^\star$), the secant method converges superlinearly. We omit the precise statement and the proof.

Exercises

12.1 The nonlinear resistive circuit shown below is described by the nonlinear equation

$$f(x) = g(x) - (E - x)/R = 0.$$

The function $g(x)$ gives the current through the nonlinear resistor as a function of the voltage x across its terminals.



Use Newton's method to find all the solutions of this nonlinear equation, assuming that

$$g(x) = x^3 - 6x^2 + 10x.$$

Consider three cases:

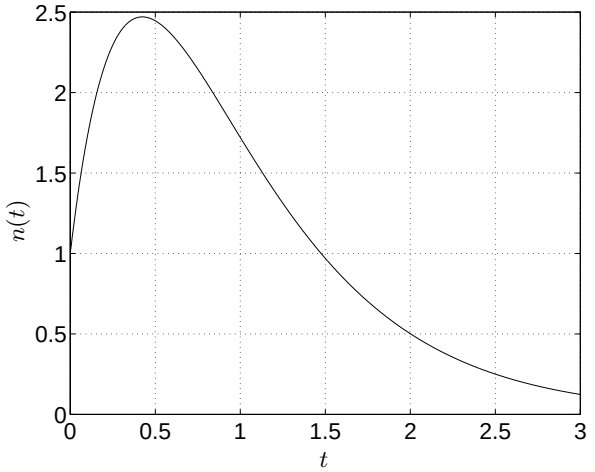
(a) $E = 5$, $R = 1$.
(b) $E = 15$, $R = 3$.
(c) $E = 4$, $R = 0.5$.

Select suitable starting points by plotting f over the interval $[0, 4]$, and visually selecting a good starting point. You can terminate the Newton iteration when $|f(x^{(k)})| < 10^{-8}$. Compare the speed of convergence for the three problems, by plotting $|f(x^{(k)})|$ versus k, or by printing out $|f(x^{(k)})|$ at each iteration.

12.2 The concentration of a certain chemical in a reaction is given by the following function of time:

$$n(t) = 10te^{-2t} + e^{-t},$$

which is shown below.



Use Newton's method to answer the following questions.

(a) Find the two values of t at which $n(t) = 2$.

(b) Find the value of t at which $n(t)$ reaches its maximum.

12.3 Find all values of x for which

$$\|(A + xI)^{-1}b\| = 1$$

where

$$A = \begin{bmatrix} -3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Use Newton's method, applied to the equation $g(x) = 1$, where

$$g(x) = \|(A + xI)^{-1}b\|^2 = \frac{1}{(-3+x)^2} + \frac{1}{(1+x)^2} + \frac{1}{(2+x)^2}.$$

First plot $g(x)$ versus x to determine the number of solutions and to select good starting points.

12.4 *A hybrid of the bisection method and Newton's method.* Newton's method for solving a nonlinear equation in one variable works well when started near a solution, but may not converge otherwise. The bisection method on the other hand does not require a good initial guess of the solution, but is much slower. In this problem you are asked to write a MATLAB implementation of a method that combines the advantages of both methods.

The algorithm is outlined as follows.

given tolerances $\epsilon_1 > 0$, $\epsilon_2 > 0$; an initial interval $[l, u]$ with $f(l)f(u) < 0$.
$x := u$.
while ($|f(x)| > \epsilon_1$ and $u - l > \epsilon_2$)
  $x_{\rm nt} := x - f(x)/f'(x)$.
  if ($l < x_{\rm nt} < u$ and $|f(x_{\rm nt})| \leq 0.99|f(x)|$), $x := x_{\rm nt}$, **else** $x := (l + u)/2$.
  if ($f(l)f(x) < 0$), $u := x$, **else**, $l := x$.

As in the bisection method, we maintain an interval $[l, u]$ for which $f(l)f(u) < 0$, so it certainly contains at least one solution. At each iteration we calculate the Newton iterate $x_{\rm nt} = x - f(x)/f'(x)$, but we only accept it as our next value of x if it lies in the interval (l, u) and it reduces $|f(x)|$ by at least 1%. If $x_{\rm nt}$ does not satisfy these two conditions, we do not use it and instead take the mid point $(u + l)/2$ as the next iterate x. We then update l and u in such a way that $f(l)f(u) < 0$ is satisfied.

Write an m-file `myfzero.m` that implements this algorithm. The calling sequence is

```
x = myfzero(fname,l,u);
```

The arguments l and u specify the initial interval and must satisfy $f(l)f(u) < 0$. The first argument `fname` is a string with the name of a function that evaluates $f(x)$ and $f'(x)$ at a given point x. It should be written in such a way that you can obtain $f(x)$ and $f'(x)$ at a given x using the MATLAB command

```
[f,fprime] = feval(fname,x);
```

(see `help feval` in MATLAB for details on `feval`). For example, in order to use `myfzero.m` to find a zero of $f(x) = \sin x$, you would have to write a second m-file `fct.m` (or any other name you like) with the following contents

```
function [f,g] = fct(x);
  f = sin(x);
  g = cos(x);
```

and then call `myfzero.m` in MATLAB using the command

```
x = myfzero('fct',-1,1);
```

(assuming you take $l = -1$, $u = 1$).
Test your code on the three functions

$$f(x) = e^x - e^{-x}, \qquad f(x) = e^x - e^{-x} - 3x \qquad f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

For each function, try a number of different starting intervals to make sure the code works correctly. Plot the values of $|f(x^{(k)})|$ and $|x^{(k)} - x^\star|$ versus the iteration number k (using `semilogy`). Attach your MATLAB code and a few typical plots.
More details.

- You can use $\epsilon_1 = 10^{-5}$ and $\epsilon_2 = 10^{-5}$. As in all iterative algorithms, it is useful to add an upper bound on the number of iterations.
- For the purpose of producing the convergence plots, you can modify `myfzero.m` so it returns an array with all values $x^{(k)}$ and $f(x^{(k)})$, not just the last iterate x.
- Avoid evaluating $f(x)$ more than once at the same point. For example, the outline above suggests that if we accept $x_{\rm nt}$ as our next iterate, we actually evaluate $f(x_{\rm nt})$ twice, the first time when we compare $|f(x_{\rm nt})|$ and $|f(x)|$ and the second time when we compare $f(l)$ with $f(x_{\rm nt})$. Function evaluations may be expensive, so you should try to avoid unnecessary function evaluations by storing function values that may be needed again later.
- The algorithm we outlined does not verify whether $f'(x) = 0$ before it calculates the Newton step. To avoid dividing by zero, you can first check if $f'(x)$ is very small. If it is, you don't compute the Newton step, but go directly to the bisection.
- When the code works properly, you expect that it switches to the Newton step eventually, when it gets close enough to a solution. While developing your code, it is therefore useful to make it display at each iteration whether it used the Newton step or the bisection step (using the `disp` command).

12.5 Use Newton's method to find all solutions of the two equations

$$(x_1 - 1)^2 + 2(x_2 - 1)^2 = 1, \qquad 3(x_1 + x_2 - 2)^2 + (x_1 - x_2 - 1)^2 = 2$$

in the two variables x_1, x_2. Each equation defines an ellipse in the (x_1, x_2)-plane. You are asked to find the points where the ellipses intersect.

12.6 Explain how you would solve the following problem using Newton's method. We are given three distinct points $a = (a_1, a_2)$, $b = (b_1, b_2)$, $c = (c_1, c_2)$ in a plane, and two positive numbers α and β. Find a point $x = (x_1, x_2)$ that satisfies

$$\|x - a\| = \alpha\|x - b\|, \qquad \|x - a\| = \beta\|x - c\|. \tag{12.6}$$

Clearly state the equations $f(x) = 0$ to which you apply Newton's method (these equations can be the two equations (12.6) or an equivalent set of equations), and the linear equations you solve at each iteration of the algorithm. You do not have to discuss the selection of the starting point, the stopping criterion, or the convergence of the method.

12.7 Very large sets of linear equations

$$Ax = b \tag{12.7}$$

are sometimes solved using *iterative* methods instead of the standard non-iterative methods (based on LU factorization), which can be too expensive or require too much memory when the dimension of A is large. A simple iterative method works as follows. Suppose A is an $n \times n$ matrix with nonzero diagonal elements. We write A as

$$A = D - B$$

where D is the diagonal part of A (a diagonal matrix with diagonal elements $D_{ii} = A_{ii}$), and $B = D - A$. The equation $Ax = b$ is then equivalent to $Dx = Bx + b$, or

$$x = D^{-1}(Bx + b).$$

The iterative algorithm consists in running the iteration

$$x^{(k+1)} = D^{-1}(Bx^{(k)} + b), \tag{12.8}$$

starting at some initial guess $x^{(0)}$. The iteration (12.8) is very cheap if the matrix B is sparse, so if the iteration converges quickly, this method may be faster than the standard LU factorization method.

Show that if $\|D^{-1}B\| < 1$, then the sequence $x^{(k)}$ converges to the solution $x = A^{-1}b$.

Chapter 13

Unconstrained minimization

13.1 Terminology

Suppose $g : \mathbf{R}^n \to \mathbf{R}$ is a scalar-valued function of n variables $x = (x_1, x_2, \ldots, x_n)$. We say $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$ *minimizes* g if $g(\hat{x}) \leq g(x)$ for all $x \in \mathbf{R}^n$. We use the notation

$$\text{minimize} \quad g(x)$$

to denote the problem of finding an $\hat{x}$ that minimizes g. This is called an *unconstrained minimization problem*, with variables x_1, ..., x_n, and with *objective function* or *cost function* g.

If $\hat{x}$ minimizes g, then we say $\hat{x}$ is a *solution* of the minimization problem, or a *minimum* of g. A minimum is also sometimes referred to as a *global* minimum. A vector $\hat{x}$ is a *local minimum* if there exists an $R > 0$ such that $g(\hat{x}) \leq g(x)$ for all x with $\|x - \hat{x}\| \leq R$. In other words, there is a neighborhood around $\hat{x}$ in which $g(x) \geq g(\hat{x})$. In this chapter, minimum means global minimum. The word 'global' in 'global minimum' is redundant, but is sometimes added for emphasis.

The greatest α such that $\alpha \leq g(x)$ for all x is called the *optimal value* of the minimization problem, and denoted

$$\min g(x)$$

or $\min_x g(x)$. If $\hat{x}$ is a minimum of g, then $g(\hat{x}) = \min g(x)$, and we say that the optimal value is *attained* at $\hat{x}$. It is possible that $\min g(x)$ is finite, but there is no $\hat{x}$ with $g(\hat{x}) = \min g(x)$ (see the examples below). In that case the optimal value is not attained. It is also possible that $g(x)$ is unbounded below, in which case we define the optimal value as $\min g(x) = -\infty$.

Examples We illustrate the definitions with a few examples with one variable.

- $g(x) = (x-1)^2$. The optimal value is $\min g(x) = 0$, and is attained at the (global) minimum $\hat{x} = 1$. There are no other local minima.
- $g(x) = e^x + e^{-x} - 3x^2$, shown in the left-hand plot of figure 13.1. The optimal value is -7.02. There are two (global) minima, at $\hat{x} = \pm 2.84$. There are no other local minima.

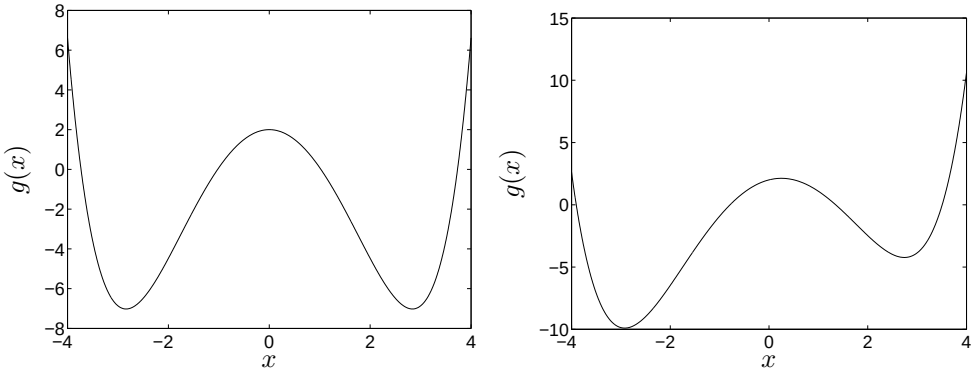


Figure 13.1 Left: the function $g(x) = e^x + e^{-x} - 3x^2$. Right: the function $g(x) = e^x + e^{-x} - 3x^2 + x$.

- $g(x) = e^x + e^{-x} - 3x^2 + x$, shown in figure 13.1 (right). The optimal value is -9.90, attained at the minimum $\hat{x} = -2.92$. There is another local minimum at $x = 2.74$.
- $g(x) = e^{-x}$. The optimal value is $\min g(x) = 0$, but is not attained. There are no local or global minima.
- $g(x) = -x + e^{-x}$. This function is unbounded below; the optimal value is $\min g(x) = -\infty$. There are no local or global minima.

13.2 Gradient and Hessian

Gradient The *gradient* of a function $g(x)$ of n variables, at $\hat{x}$, is the vector of first partial derivatives evaluated at $\hat{x}$, and is denoted $\nabla g(\hat{x})$:

$$\nabla g(\hat{x}) = \begin{bmatrix} \partial g(\hat{x})/\partial x_1 \\ \partial g(\hat{x})/\partial x_2 \\ \vdots \\ \partial g(\hat{x})/\partial x_n \end{bmatrix}.$$

As mentioned in chapter 1, the gradient is used in the first-order (or affine) approximation of g around $\hat{x}$,

$$\begin{aligned} g_{\mathrm{aff}}(x) &= g(\hat{x}) + \sum_{i=1}^{n} \frac{\partial g(\hat{x})}{\partial x_i}(x_i - \hat{x}_i) \\ &= g(\hat{x}) + \nabla g(\hat{x})^T (x - \hat{x}). \end{aligned}$$

If $n = 1$, the gradient is simply the first derivative $g'(\hat{x})$, and the first-order approximation reduces to

$$g_{\rm aff}(x) = g(\hat{x}) + g'(\hat{x})(x - \hat{x}).$$

Hessian The *Hessian* of a function $g(x)$ of n variables, at $\hat{x}$, is the matrix of second partial derivatives evaluated at $\hat{x}$, and is denoted as $\nabla^2 g(\hat{x})$:

$$\nabla^2 g(\hat{x}) = \begin{bmatrix} \partial^2 g(\hat{x})/\partial x_1^2 & \partial^2 g(\hat{x})/\partial x_1 \partial x_2 & \cdots & \partial^2 g(\hat{x})/\partial x_1 \partial x_n \\ \partial^2 g(\hat{x})/\partial x_2 \partial x_1 & \partial^2 g(\hat{x})/\partial x_2^2 & \cdots & \partial^2 g(\hat{x})/\partial x_2 \partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2 g(\hat{x})/\partial x_n \partial x_1 & \partial^2 g(\hat{x})/\partial x_n \partial x_2 & \cdots & \partial^2 g(\hat{x})/\partial x_n^2 \end{bmatrix}.$$

This is a symmetric matrix, because $\partial^2 g(\hat{x})/\partial x_i \partial x_j = \partial^2 g(\hat{x})/\partial x_j \partial x_i$.

The Hessian is related to the second-order (or quadratic) approximation of g around $\hat{x}$, which is defined as

$$\begin{aligned} g_{\rm quad}(x) &= g(\hat{x}) + \sum_{i=1}^n \frac{\partial g(\hat{x})}{\partial x_i}(x_i - \hat{x}_i) + \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \frac{\partial^2 g(\hat{x})}{\partial x_i \partial x_j}(x_i - \hat{x}_i)(x_j - \hat{x}_j) \\ &= g(\hat{x}) + \nabla g(\hat{x})^T (x - \hat{x}) + \frac{1}{2}(x - \hat{x})^T \nabla^2 g(\hat{x})(x - \hat{x}). \end{aligned}$$

If $n = 1$, the Hessian is the second derivative $g''(\hat{x})$, and the second-order approximation reduces to

$$g_{\rm quad}(x) = g(\hat{x}) + g'(\hat{x})(x - \hat{x}) + \frac{1}{2} g''(\hat{x})(x - \hat{x})^2.$$

As an example, the Hessian of the function g in (1.6) is

$$\nabla^2 g(x) = \begin{bmatrix} e^{x_1 + x_2 - 1} + e^{x_1 - x_2 - 1} + e^{-x_1 - 1} & e^{x_1 + x_2 - 1} - e^{x_1 - x_2 - 1} \\ e^{x_1 + x_2 - 1} - e^{x_1 - x_2 - 1} & e^{x_1 + x_2 - 1} + e^{x_1 - x_2 - 1} \end{bmatrix}, \qquad (13.1)$$

so the second-order approximation around $\hat{x} = 0$ is

$$g_{\rm quad}(x) = \frac{1}{e}(3 + x_1 + (3/2)x_1^2 + x_2^2).$$

Properties We list here a few properties that often simplify the task of calculating gradients and Hessians. These facts are straightforward (although sometimes tedious) to verify, directly from the definition of gradient and Hessian.

1. *Linear and affine functions.* The gradient and Hessian of $g(x) = a^T x + b$ are

$$\nabla g(x) = a, \qquad \nabla^2 g(x) = 0.$$

2. *Quadratic functions.* The gradient and Hessian of $g(x) = x^T P x + q^T x + r$, where P is a symmetric matrix, are

$$\nabla g(x) = 2Px + q, \qquad \nabla^2 g(x) = 2P.$$

3. *Sum of two functions.* If $g(x) = g_1(x) + g_2(x)$, then

$$\nabla g(x) = \nabla g_1(x) + \nabla g_2(x), \qquad \nabla^2 g(x) = \nabla^2 g_1(x) + \nabla^2 g_2(x).$$

4. *Scalar multiplication.* If $g(x) = \alpha f(x)$ where α is a scalar, then

$$\nabla g(x) = \alpha \nabla f(x), \qquad \nabla^2 g(x) = \alpha \nabla^2 f(x).$$

5. *Composition.* If $g(x) = f(Cx + d)$ where C is an $m \times n$-matrix, d is an m-vector, and $f(y)$ is a function of m variables, then

$$\nabla g(x) = C^T \nabla f(Cx + d), \qquad \nabla^2 g(x) = C^T \nabla^2 f(Cx + d) C.$$

   Note that $f(Cx + d)$ denotes the function $f(y)$, evaluated at $y = Cx + d$. Similarly, $\nabla f(Cx + d)$ is the gradient $\nabla f(y)$, evaluated at $y = Cx + d$, and $\nabla^2 f(Cx + d)$ is the Hessian $\nabla^2 f(y)$, evaluated at $y = Cx + d$.

Examples As a first example, consider the least-squares function

$$g(x) = \|Ax - b\|^2.$$

We can find the gradient and Hessian by expanding g in terms of its variables x_i, and then taking the partial derivatives. An easier derivation is from the properties listed above. We can express g as

$$\begin{aligned} g(x) &= (Ax - b)^T (Ax - b) \\ &= x^T A^T A x - b^T A x - x^T A^T b + b^T b \\ &= x^T A^T A x - 2 b^T A x + b^T b. \end{aligned}$$

This shows that g is a quadratic function: $g(x) = x^T P x + q^T x + r$ with $P = A^T A$, $q = -2A^T b$, $r = b^T b$. From property 2,

$$\nabla g(x) = 2A^T A x - 2A^T b, \qquad \nabla^2 g(x) = 2A^T A.$$

An alternative derivation is based on property 5. We can express g as $g(x) = f(Cx + d)$ where $C = A$, $d = -b$, and

$$f(y) = \|y\|^2 = \sum_{i=1}^m y_i^2.$$

The gradient and Hessian of f are

$$\nabla f(y) = \begin{bmatrix} 2y_1 \\ 2y_2 \\ \vdots \\ 2y_m \end{bmatrix} = 2y, \qquad \nabla^2 f(y) = \begin{bmatrix} 2 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 2 \end{bmatrix} = 2I.$$

Applying property 5, we find that

$$\begin{aligned}\nabla g(x) &= A^T\nabla f(Ax-b)\\ &= 2A^T(Ax-b),\\ \nabla^2 g(x) &= A^T\nabla^2 f(Ax-b)A\\ &= 2A^TA,\end{aligned}$$

the same expressions as we derived before.

We can use the same method to find the gradient and Hessian of the function in (1.6),

$$g(x_1,x_2) = e^{x_1+x_2-1} + e^{x_1-x_2-1} + e^{-x_1-1}.$$

We can express g as $g(x) = f(Cx+d)$, where $f(y) = e^{y_1}+e^{y_2}+e^{y_3}$, and

$$C = \begin{bmatrix} 1 & 1\\ 1 & -1\\ -1 & 0\end{bmatrix}, \qquad d = \begin{bmatrix}-1\\-1\\-1\end{bmatrix}.$$

The gradient and Hessian of f are

$$\nabla f(y) = \begin{bmatrix} e^{y_1}\\ e^{y_2}\\ e^{y_3}\end{bmatrix}, \qquad \nabla^2 f(y) = \begin{bmatrix} e^{y_1} & 0 & 0\\ 0 & e^{y_2} & 0\\ 0 & 0 & e^{y_3}\end{bmatrix},$$

so it follows from property 5 that

$$\nabla g(x) = C^T\nabla f(Cx+d) = \begin{bmatrix} 1 & 1 & -1\\ 1 & -1 & 0\end{bmatrix}\begin{bmatrix} e^{x_1+x_2-1}\\ e^{x_1-x_2-1}\\ e^{-x_1-1}\end{bmatrix}$$

and

$$\begin{aligned}\nabla^2 g(x) &= C^T\nabla^2 f(Cx+d)C \qquad (13.2)\\ &= \begin{bmatrix} 1 & 1 & -1\\ 1 & -1 & 0\end{bmatrix}\begin{bmatrix} e^{x_1+x_2-1} & 0 & 0\\ 0 & e^{x_1-x_2-1} & 0\\ 0 & 0 & e^{-x_1-1}\end{bmatrix}\begin{bmatrix} 1 & 1\\ 1 & -1\\ -1 & 0\end{bmatrix}.\end{aligned}$$

It can be verified that these expressions are the same as (1.7) and (13.1).

13.3 Optimality conditions

Local or global minima of a function g can be characterized in terms of the gradient and Hessian. In this section we state the optimality conditions (without proofs).

Global optimality A function g is *convex* if $\nabla^2 g(x)$ is positive semidefinite everywhere (for all x). If g is convex, then $x^\star$ is a minimum if and only if

$$\nabla g(x^\star) = 0.$$

There are no other local minima, *i.e.*, every local minimum is global.

Examples with $n = 1$ A function of one variable is convex if $g''(x) \geq 0$ everywhere. Therefore $x^\star$ is a minimum of a convex function if and only if

$$g'(x^\star) = 0.$$

The functions $g(x) = x^2$ and $g(x) = x^4$ are convex, with second derivatives $g''(x) = 2$ and $g''(x) = 12x^2$, respectively. Therefore we can find the minimum by setting the first derivative equal to zero, which in both cases yields $x^\star = 0$.

The first and second derivatives of the function

$$g(x) = \log(e^x + e^{-x})$$

are

$$g'(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad g''(x) = \frac{4}{(e^x + e^{-x})^2}.$$

The second derivative is nonnegative everywhere, so g is convex, and we can find its minimum by solving $g'(x^\star) = 0$, which gives $x^\star = 0$.

Examples with $n > 1$ A quadratic function $g(x) = x^T Px + q^T x + r$ (with P symmetric) is convex if P is positive semidefinite. (Recall that $\nabla^2 g(x) = 2P$.) Therefore $x^\star$ is a minimum if and only if

$$\nabla g(x^\star) = 2Px^\star + q = 0,$$

which is a set of n linear equations in n variables. If P is positive definite, the equations have a unique solution $x^\star = -(1/2)P^{-1}q$, and can be efficiently solved using the Cholesky factorization.

The least-squares function $g(x) = \|Ax-b\|^2$ is convex, because $\nabla^2 g(x) = 2A^T A$, and the matrix $A^T A$ is positive semidefinite. Therefore $x^\star$ is a minimum if and only if

$$\nabla g(x^\star) = 2A^T Ax^\star - 2A^T b = 0.$$

We can find $x^\star$ by solving the set of linear equations

$$A^T Ax^\star = A^T b,$$

in which we recognize the normal equations associated with the least-squares problem.

In these first two examples, we can solve $\nabla g(x^\star) = 0$ by solving a set of linear equations, so we do not need an iterative method to minimize g. In general, however, the optimality condition $\nabla g(x^\star) = 0$ is a set of nonlinear equations in $x^\star$, which have to be solved by an iterative algorithm.

The function g defined in (1.6), for example, is convex, because its Hessian is positive definite everywhere. Although that is not obvious from the expression in (13.1), it follows from (13.2) which shows that

$$\nabla^2 g(x) = C^T DC$$

where D is a diagonal matrix with diagonal elements

$$d_{11} = e^{x_1+x_1-1}, \qquad d_{22} = e^{x_1+x_1-1}, \qquad d_{33} = e^{-x_1-1},$$

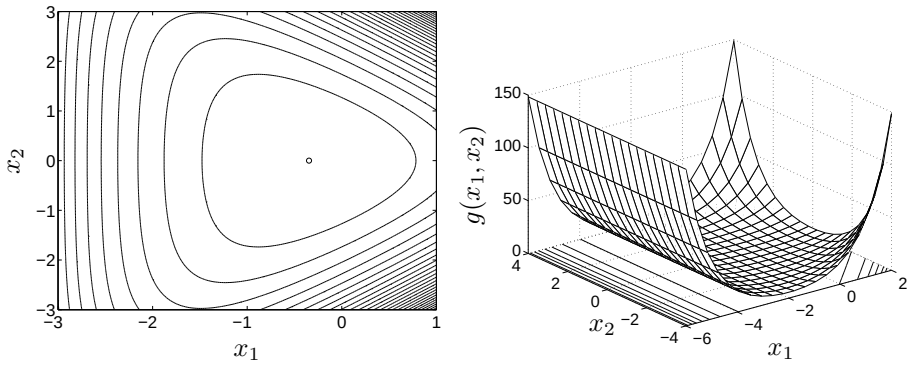


Figure 13.2 Contour lines and graph of the function $g(x) = \exp(x_1 + x_2 - 1) + \exp(x_1 - x_2 - 1) + \exp(-x_1 - 1)$.

and C is a left-invertible 3×2 matrix. It can be shown that $v^T C^T DCv > 0$ for all nonzero v, hence $C^T DC$ is positive definite. It follows that x is a minimum of g if and only if

$$\nabla g(x) = \begin{bmatrix} e^{x_1+x_2-1} + e^{x_1-x_2-1} - e^{-x_1-1} \\ e^{x_1+x_2-1} - e^{x_1-x_2-1} \end{bmatrix} = 0.$$

This is a set of two nonlinear equations in two variables.

The contour lines and graph of the function g are shown in figure 13.2.

Local optimality It is much harder to characterize optimality if g is not convex (*i.e.*, if there are points where the Hessian is not positive semidefinite). It is not sufficient to set the gradient equal to zero, because such a point might correspond to a local minimum, a local maximum, or a saddle point (see for example, the second function in figure 13.1). However, we can state some simple conditions for *local* optimality.

- *Necessary condition.* If $x^\star$ is locally optimal, then $\nabla g(x^\star) = 0$ and $\nabla^2 g(x^\star)$ is positive semidefinite.

- *Sufficient condition.* If $\nabla g(x^\star) = 0$ and $\nabla^2 g(x^\star)$ is positive definite, then $x^\star$ is locally optimal.

The function $g(x) = x^3$ provides an example that shows that 'positive definite' cannot be replaced by 'positive semidefinite' in the sufficient condition. At $x = 0$ it satisfies $g'(x) = 3x^2 = 0$ and $g''(x) = 6x = 0$, although $x = 0$ is not a local minimum.

13.4 Newton's method for minimizing a convex function

We first consider the important case when the objective function g is convex. As we have seen in section 13.3, we can find the minimum by solving $\nabla g(x) = 0$. This is a set of n nonlinear equations in n variables, that we can solve using any method for nonlinear equations, for example, Newton's method.

For simplicity we assume that $\nabla^2 g(x)$ is positive definite everywhere, which is a little stronger than requiring $\nabla^2 g(x)$ to be positive semidefinite.

Newton's method for solving nonlinear equations, applied to $\nabla g(x) = 0$, is based on the iteration

$$x^{(k+1)} = x^{(k)} - \nabla^2 g(x^{(k)})^{-1} \nabla g(x^{(k)}), \quad k = 0, 1, 2, \ldots.$$

A more detailed description is as follows.

Algorithm 13.1. Newton's method for unconstrained minimization.

given initial x, tolerance $\epsilon > 0$
repeat

1. Evaluate $\nabla g(x)$ and $\nabla^2 g(x)$.
2. **if** $\|\nabla g(x)\| \leq \epsilon$, **return** x.
3. Solve $\nabla^2 g(x) v = -\nabla g(x)$.
4. $x := x + v$.

until a limit on the number of iterations is exceeded

Since $\nabla^2 g(x)$ is positive definite, we can use the Cholesky factorization in step 3.

The vector v computed in the kth iteration is called the *Newton step* at $x^{(k)}$:

$$v^{(k)} = -\nabla^2 g(x^{(k)})^{-1} \nabla g(x^{(k)}).$$

The Newton step can be interpreted in several ways.

Interpretation as solution of linearized optimality condition In chapter 12 we have seen that the iterates in Newton's method for solving nonlinear equations can be interpreted as solutions of linearized problems.

If we linearize the optimality condition $\nabla g(x) = 0$ near $\hat{x} = x^{(k)}$ we obtain

$$\nabla g(x) \approx \nabla g(\hat{x}) + \nabla^2 g(\hat{x})(x - \hat{x}) = 0.$$

This is a linear equation in x, with solution

$$x = \hat{x} - \nabla^2 g(\hat{x})^{-1} \nabla g(\hat{x}) = x^{(k)} + v^{(k)}.$$

So the Newton step $v^{(k)}$ is what must be added to $x^{(k)}$ so that the linearized optimality condition holds.

When $n = 1$ this interpretation is particularly simple. The solution of the linearized optimality condition is the zero-crossing of the derivative $g'(x)$, which is monotonically increasing since $g''(x) > 0$. Given our current approximation $x^{(k)}$ of the solution, we form a first-order Taylor approximation of $g'(x)$ at $x^{(k)}$. The zero-crossing of this approximation is then $x^{(k)} + v^{(k)}$. This interpretation is illustrated in figure 13.3.

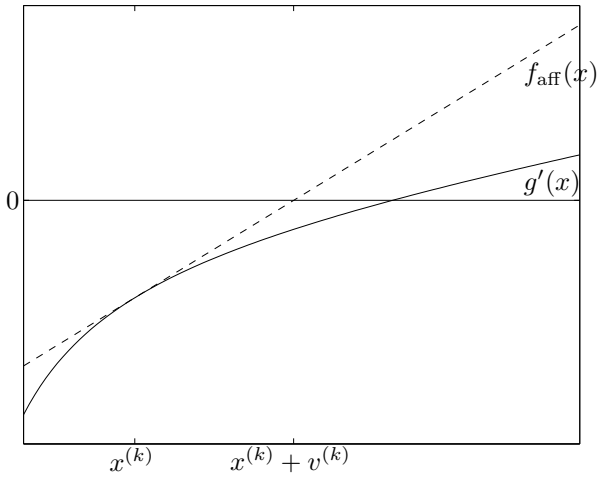


Figure 13.3 The solid curve is the derivative $g'(x)$ of the function g. $f_{\text{aff}}(x) = g'(x^{(k)}) + g''(x^{(k)})(x - x^{(k)})$ is the affine approximation of $g'(x)$ at $x^{(k)}$. The Newton step $v^{(k)}$ is the difference between the root of f_{aff} and the point $x^{(k)}$.

Interpretation as minimum of second-order approximation The second-order approximation g_{quad} of g near $\hat{x} = x^{(k)}$ is

$$g_{\text{quad}}(x) = g(\hat{x}) + \nabla g(\hat{x})^T(x - \hat{x}) + \frac{1}{2}(x - \hat{x})^T \nabla^2 g(\hat{x})(x - \hat{x}) \tag{13.3}$$

which is a convex quadratic function of y. To find the gradient and Hessian of g_{quad}, we express the function as

$$g_{\text{quad}}(x) = x^T P x + q^T x + r$$

where

$$P = \frac{1}{2}\nabla^2 g(\hat{x}), \qquad q = \nabla g(\hat{x}) - \nabla^2 g(\hat{x})\hat{x}, \qquad r = g(\hat{x}) - \nabla g(\hat{x})^T \hat{x} + \frac{1}{2}\hat{x}^T \nabla^2 g(\hat{x})\hat{x},$$

and then apply the second property in section 13.2:

$$\nabla^2 g_{\text{quad}}(x) = 2P = \nabla^2 g(\hat{x})$$

and

$$\nabla g_{\text{quad}}(x) = 2Px + q = \nabla g(\hat{x}) + \nabla^2 g(\hat{x})(x - \hat{x}).$$

The minimum of g_{quad} is

$$x = -\frac{1}{2}P^{-1}q = \hat{x} - \nabla^2 g(\hat{x})^{-1}\nabla g(\hat{x}).$$

Thus, the Newton step $v^{(k)}$ is what should be added to $x^{(k)}$ to minimize the second-order approximation of g at $x^{(k)}$. This is illustrated in figure 13.4.

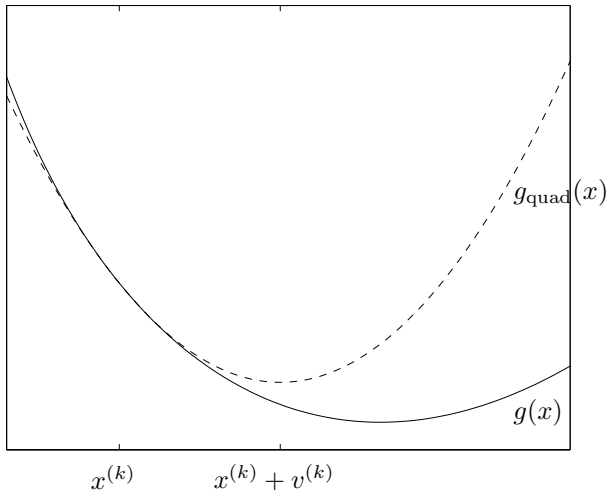


Figure 13.4 The function g (shown solid) and its second-order approximation g_{quad} at $x^{(k)}$ (dashed). The Newton step $v^{(k)}$ is the difference between the minimum of g_{quad} and the point $x^{(k)}$.

This interpretation gives us some insight into the Newton step. If the function g is quadratic, then $x^{(k)} + v^{(k)}$ is the exact minimum of g. If the function g is nearly quadratic, intuition suggests that $x^{(k)} + v^{(k)}$ should be a very good estimate of the minimum of g. The quadratic model of f will be very accurate when $x^{(k)}$ is near $x^\star$. It follows that when $x^{(k)}$ is near $x^\star$, the point $x^{(k)} + v^{(k)}$ should be a very good estimate of $x^\star$.

Example We apply Newton's method to

$$g(x) = \log(e^x + e^{-x}).$$

Figure 13.5 shows the iteration when we start at $x^{(0)} = 0.85$. Figure 13.6 shows the iteration when we start at $x^{(0)} = 1.15$.

We notice that Newton's method only converges when started near the solution, as expected based on the general properties of the method. In the next paragraph we describe a simple and easily implemented modification that makes the method globally convergent (*i.e.*, convergent from any starting point).

13.5 Newton's method with backtracking

The purpose of backtracking is to avoid the behavior of figure 13.6, in which the function values *increase* from iteration to iteration. A closer look at the example shows that there is actually nothing wrong with the direction of the Newton step, since it always points in the direction of decreasing g. The problem is that we step

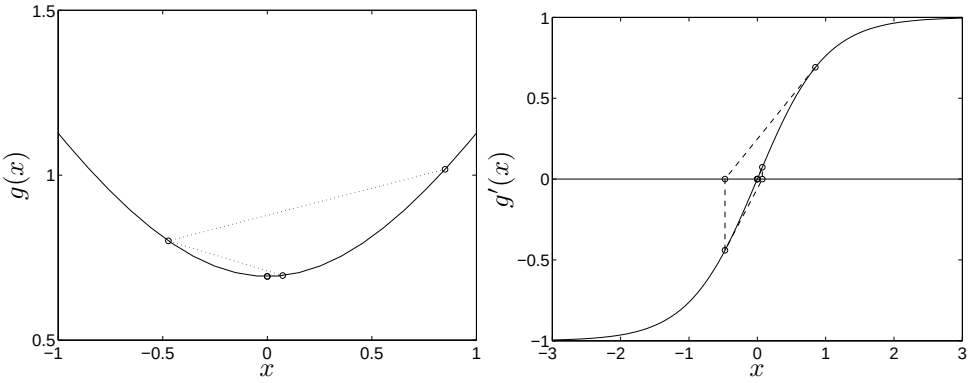


Figure 13.5 The solid line in the left figure is $g(x) = \log(\exp(x) + \exp(-x))$. The circles indicate the function values at the successive iterates in Newton's method, starting at $x^{(0)} = 0.85$. The solid line in the right figure is the derivative $g'(x)$. The dashed lines in the right-hand figure illustrate the first interpretation of Newton's method (linearization of the optimality condition $g'(x) = 0$).

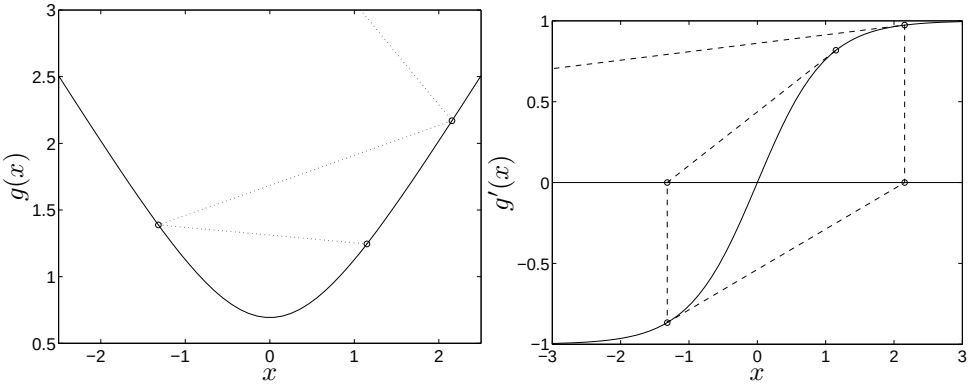


Figure 13.6 The solid line in the left figure is $g(x) = \log(\exp(x) + \exp(-x))$. The circles indicate the function values at the successive iterates in Newton's method, starting at $x^{(0)} = 1.15$. The solid line in the right figure is the derivative $g'(x)$. The dashed lines in the right-hand figure illustrate the first interpretation of Newton's method.

too far in that direction. So the remedy is quite obvious. At each iteration, we first attempt the full Newton step $x^{(k)} + v^{(k)}$, and evaluate g at that point. If the function value $g(x^{(k)} + v^{(k)})$ is higher than $g(x^{(k)})$, we reject the update, and try $x^{(k)} + (1/2)v^{(k)}$, instead. If the function value is still higher than $g(x^{(k)})$, we try $x^{(k)} + (1/4)v^{(k)}$, and so on, until a value of t is found with $g(x^{(k)} + tv^{(k)}) < g(x^{(k)})$. We then take $x^{(k+1)} = x^{(k)} + tv^{(k)}$.

In practice, the backtracking idea is often implemented as shown in the following outline.

Algorithm 13.2. NEWTON'S METHOD WITH BACKTRACKING.

given initial x, tolerance $\epsilon > 0$, parameter $\alpha \in (0, 1/2)$.
repeat

1. Evaluate $\nabla g(x)$ and $\nabla^2 g(x)$.
2. **if** $\|\nabla g(x)\| \leq \epsilon$, **return** x.
3. Solve $\nabla^2 g(x)v = -\nabla g(x)$.
4. $t := 1$.
   while $g(x + tv) > g(x) + \alpha t \nabla g(x)^T v$, $t := t/2$.
5. $x := x + tv$.

until a limit on the number of iterations is exceeded

The parameter α is usually chosen quite small (*e.g.*, $\alpha = 0.01$).

The scalar t computed in step 4 is called the *step size*, and the algorithm used to calculate the step size (*i.e.*, step 4) is called the *line search*. During the line search, we examine candidate updates $x + tv$ (for $t = 1, 1/2, 1/4, \ldots$) on the line that passes through x and $x + v$, hence the name line search.

The purpose of the line search is to find a step size t such that $g(x + tv)$ is sufficiently less than $g(x)$. More specifically, the condition of sufficient decrease (used in step 4) is that the step size t is accepted if

$$g(x + tv) \leq g(x) + \alpha t \nabla g(x)^T v. \tag{13.4}$$

To clarify this condition, we consider $g(x + tv)$, where $x = x^{(k)}$, $v = v^{(k)}$, as a function of t. The function $h(t) = g(x + tv)$ is a function of one variable t, and gives the values of g on the line $x + tv$, as a function of the step size t (see figure 13.7). At $t = 0$, we have $h(0) = g(x)$.

A first important observation is that the quantity $\nabla g(x)^T v$ is the derivative of h at $t = 0$. More generally,

$$\begin{aligned} h'(t) &= \frac{\partial g(x+tv)}{\partial x_1} v_1 + \frac{\partial g(x+tv)}{\partial x_2} v_2 + \cdots + \frac{\partial g(x+tv)}{\partial x_n} v_n \\ &= \nabla g(x + tv)^T v, \end{aligned}$$

so at $t = 0$ we have $h'(0) = \nabla g(x)^T v$. It immediately follows that $h'(0) < 0$:

$$h'(0) = \nabla g(x)^T v = -v^T \nabla^2 g(x) v < 0,$$

because v is defined as $v = -\nabla^2 g(x)^{-1} \nabla g(x)$, and $\nabla^2 g(x)$ is positive definite.

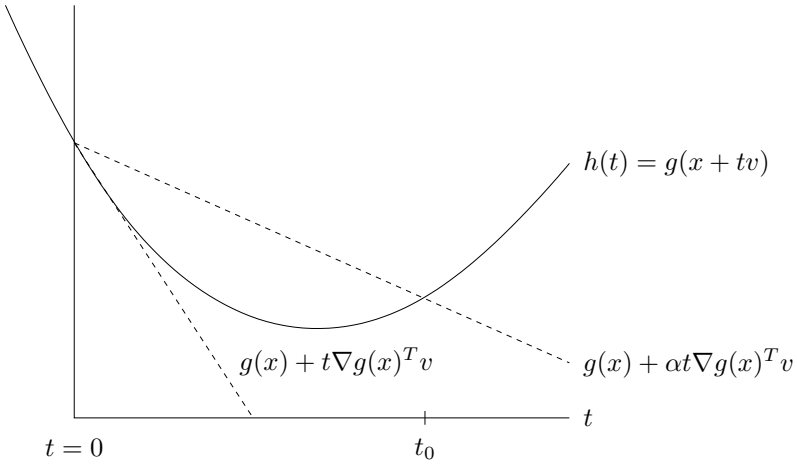


Figure 13.7 Backtracking line search. The curve shows g, restricted to the line over which we search. The lower dashed line shows the linear extrapolation of g, and the upper dashed line has a slope a factor of α smaller. The backtracking condition is that $g(x + tv)$ lies below the upper dashed line, *i.e.*, $0 \leq t \leq t_0$. The line search starts with $t = 1$, and divides t by 2 until $t \leq t_0$.

The linear approximation of $h(t) = g(x + tv)$ at $t = 0$ is

$$h(0) + h'(0)t = g(x) + t\nabla g(x)^T v,$$

so for small enough t we have

$$g(x + tv) \approx g(x) + t\nabla g(x)^T v < g(x) + \alpha t \nabla g(x)^T v.$$

This shows that the backtracking line search eventually terminates. The constant α can be interpreted as the fraction of the decrease in g predicted by linear extrapolation that we will accept.

Examples We start with two small examples. Figure 13.8 shows the iterations in Newton's method with backtracking, applied to the example of figure 13.6, starting from $x^{(0)} = 4$. As expected the convergence problem has been resolved. From the plot of the step sizes we note that the method accepts the full Newton step ($t = 1$) after a few iterations. This means that near the solution the algorithm works like the pure Newton method, which ensures fast (quadratic) convergence.

Figure 13.9 shows the results of Newton's method applied to

$$g(x_1, x_2) = \exp(x_1 + x_2 - 1) + \exp(x_1 - x_2 - 1) + \exp(-x_1 - 1),$$

starting at $x^{(0)} = (-2, 2)$.

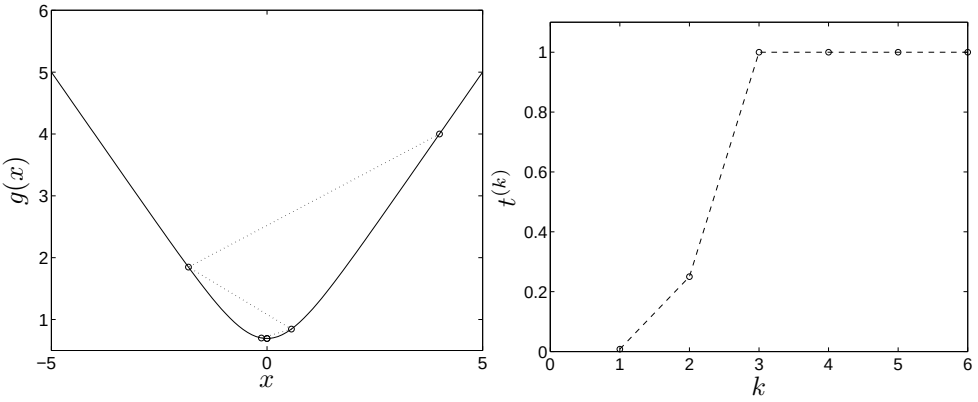


Figure 13.8 The solid line in the figure on the left is $g(x) = \log(\exp(x) + \exp(-x))$. The circles indicate the function values at the successive iterates in Newton's method with backtracking, starting at $x^{(0)} = 4$. The right-hand figure shows the step size $t^{(k)}$ at each iteration. In the first iteration ($k = 0$) the step size is $1/2^7$, *i.e.*, 7 backtracking steps were made. In the second iteration the step size is 0.25 (2 backtracking steps). The other iterations take a full Newton step ($t = 1$).

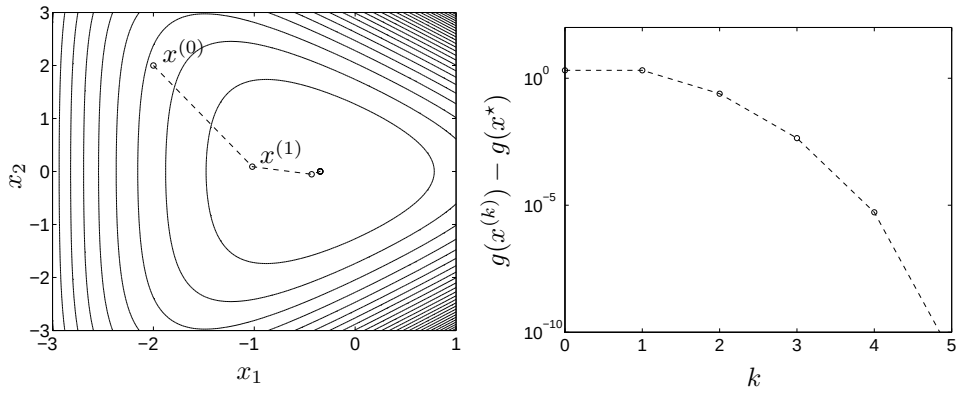


Figure 13.9 The left figure shows the contour lines of the function $g(x) = \exp(x_1 + x_2 - 1) + \exp(x_1 - x_2 - 1) + \exp(-x_1 - 1)$. The circles indicate the iterates of Newton's method, started at $(-2, 2)$. The figure on the right shows $g(x^{(k)}) - g(x^\star)$ versus k.

To give a larger example, suppose we are given an $m \times n$-matrix A and an m-vector b, and we are interested in minimizing

$$g(x) = \sum_{i=1}^{m} \log(e^{a_i^T x - b_i} + e^{-a_i^T x + b_i}). \tag{13.5}$$

where a_i^T denotes the ith row of A. To implement Newton's method, we need the gradient and Hessian of g. The easiest method is to use the composition property in section 13.2. We express g as $g(x) = f(Ax-b)$, where f is the following function of m variables:

$$f(y) = \sum_{i=1}^{m} \log(e^{y_i} + e^{-y_i}).$$

The partial derivatives of f are given by

$$\frac{\partial f(y)}{\partial y_i} = \frac{e^{y_i} - e^{-y_i}}{e^{y_i} + e^{-y_i}}, \qquad \frac{\partial^2 f(y)}{\partial y_i \partial y_j} = \begin{cases} 4/(e^{y_i} + e^{-y_i})^2 & i = j \\ 0 & i \neq j. \end{cases}$$

(In other words, $\nabla^2 f(y)$ is diagonal with diagonal elements $4/(\exp(y_i)+\exp(-y_i))^2$.) Given the gradient and Hessian of f, we can find the gradient and Hessian of g:

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

Once we have the correct expressions for the gradient and Hessian, the implementation of Newton's method is straightforward. The MATLAB code is given below.

```
x = ones(n,1);
for k=1:50
   y = A*x-b;
   val = sum(log(exp(y)+exp(-y)));
   grad = A'*((exp(y)-exp(-y))./(exp(y)+exp(-y)));
   if (norm(grad) < 1e-5), break;  end;
   hess = 4*A'*diag(1./(exp(y)+exp(-y)).^2)*A;
   v = -hess\grad;
   t = 1;
   while ( sum(log(exp(A*(x+t*v)-b)+exp(-A*(x+t*v)+b))) ...
           > val + 0.01*t*grad'*v),  t = 0.5*t;  end;
   x = x+t*v;
end;
```

We start with $x^{(0)} = (1, 1, \ldots, 1)$, set the line search parameter α to $\alpha = 0.01$, and terminate if $\|\nabla f(x)\| \leq 10^{-5}$.

The results, for an example with $m = 500$ and $n = 100$ are shown in figure 13.10. We note that many backtracking steps are needed in the first few iterations, until we reach a neighborhood of the solution in which the pure Newton method converges.

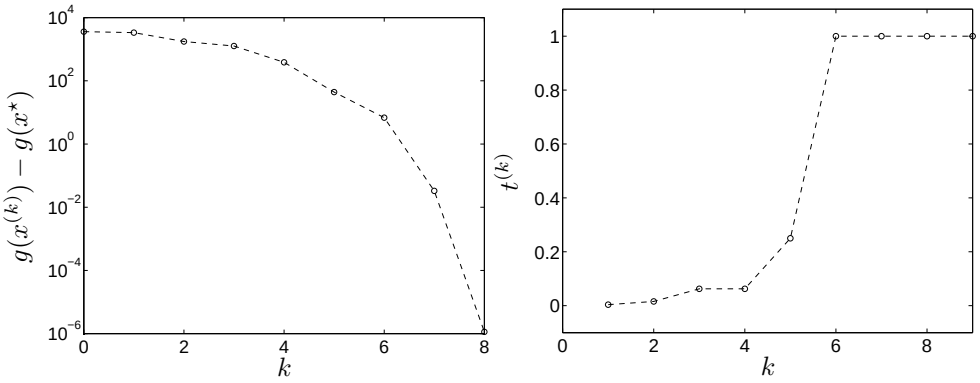


Figure 13.10 Results of Newton's method with backtracking applied to the function (13.5) for an example with $m = 500$, and $n = 100$. The figure on the left is $g(x^{(k)}) - g(x^\star)$ versus k. The convergence accelerates as k increases, and is very fast in the last few iterations. The right-hand figure shows the step size $t^{(k)}$ versus k. The last four iterations use a full Newton step ($t = 1$).

13.6 Newton's method for nonconvex functions

Minimization problems with a nonconvex cost function g present additional difficulties to Newton's method, even if the only goal is to find a local minimum. If $\nabla^2 g(x^{(k)})$ is not positive definite, then the Newton step

$$v^{(k)} = -\nabla^2 g(x^{(k)})^{-1} \nabla g(x^{(k)}),$$

might not be a descent direction. In other words, it is possible that the function $h(t) = g(x^{(k)} + tv^{(k)})$ that we considered in figure 13.7, has a *positive* slope at $t = 0$. To see this, recall that the slope of h at $t = 0$ is given by

$$h'(0) = \nabla g(x^{(k)})^T v^{(k)},$$

and that the proof that $h'(0) < 0$ in section 13.5 was based on the assumption that $\nabla^2 g(x^{(k)})$ was positive definite. Figure 13.11 shows an example in one dimension.

Practical implementations of Newton's method for nonconvex functions make sure that the direction $v^{(k)}$ used at each iteration is a descent direction, *i.e.*, a direction that satisfies

$$h'(0) = \nabla g(x^{(k)})^T v^{(k)} < 0.$$

Many techniques have been proposed to find such a $v^{(k)}$. The simplest choice is to take $v^{(k)} = -\nabla g(x^{(k)})$ at points where the Hessian is not positive definite. This is a descent direction, because

$$h'(0) = \nabla g(x^{(k)})^T v^{(k)} = -\nabla g(x^{(k)})^T \nabla g(x^{(k)}) = -\|\nabla g(x^{(k)})\|^2.$$

With this addition, Newton's method can be summarized as follows.

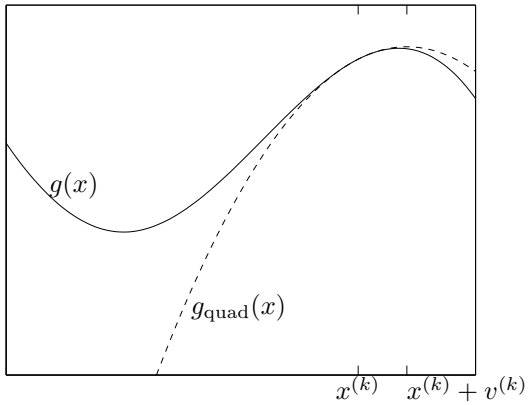


Figure 13.11 A nonconvex function $g(x)$ and the second-order approximation $g_{\text{quad}}(x)$ of g around $x^{(k)}$. The Newton step $v^{(k)} = -g'(x^{(k)})/g''(x^{(k)})$ is the difference between the maximum of g_{quad} and $x^{(k)}$.

Algorithm 13.3. NEWTON'S METHOD FOR NONCONVEX LOCAL MINIMIZATION.

given initial x, tolerance $\epsilon > 0$, parameter $\alpha \in (0, 1/2)$.
repeat

1. Evaluate $\nabla g(x)$ and $\nabla^2 g(x)$.
2. **if** $\|\nabla g(x)\| \leq \epsilon$, **return** x.
3. **if** $\nabla^2 g(x)$ is positive definite, solve $\nabla^2 g(x)v = -\nabla g(x)$ for v
   else, $v := -\nabla g(x)$.
4. $t := 1$.
   while $g(x + tv) > g(x) + \alpha t \nabla g(x)^T v$, $t := t/2$.
5. $x := x + tv$.

until a limit on the number of iterations is exceeded

Although the algorithm guarantees that the function values decrease at each iteration, it is still far from perfect. Note for example, that if we start at a point where $\nabla g(x)$ is zero (or very small), the algorithm terminates immediately in step 2, although we might be at a local maximum. Experience also shows that the convergence can be very slow, until we get close to a local minimum where the Hessian is positive definite. Practical implementations of Newton's method for nonconvex minimization use more complicated methods for finding good descent directions when the Hessian is not positive definite, and more sophisticated line searches.

Exercises

Gradient and Hessian

13.1 Derive the gradient and Hessian of the following functions g. Show that $\nabla^2 g(x)$ is positive definite.

(a) $g(x) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2 + 1} = \sqrt{\|x\|^2 + 1}$.
Hint. Express $\nabla^2 g(x)$ as

$$\nabla^2 g(x) = \frac{1}{g(x)}(I - uu^T)$$

where u is an n-vector with norm less than one. Then prove that $I - uu^T$ is positive definite.

(b) $g(x) = \sqrt{\|Cx\|^2 + 1}$ where A is a left-invertible $m \times n$ matrix.
Hint. Use the result of part (a) and the expression

$$\nabla^2 g(x) = C^T \nabla^2 h(Cx + d)C$$

for the Hessian of the function $g(x) = h(Cx + d)$.

13.2 Consider the problem of fitting a quadratic function

$$f(t) = \alpha + \beta t + \gamma t^2$$

to m given points (t_i, y_i), $i = 1, \ldots, m$. Suppose we choose to estimate the parameters α, β, γ by minimizing the function

$$g(\alpha, \beta, \gamma) = -\sum_{i=1}^{m} \log\left(1 - (\alpha + \beta t_i + \gamma t_i^2 - y_i)^2\right)$$

(where log denotes the natural logarithm).

(a) Give expressions for the gradient $\nabla g(\alpha, \beta, \gamma)$ and Hessian $\nabla^2 g(\alpha, \beta, \gamma)$ of g, at a point (α, β, γ) that satisfies

$$|\alpha + \beta t_i + \gamma t_i^2 - y_i| < 1, \quad i = 1, \ldots, m.$$

(This condition guarantees that g and its derivatives are defined at (α, β, γ).)

(b) Show that the Hessian is positive definite.

Assume that the values t_i are distinct and that $m \geq 3$.

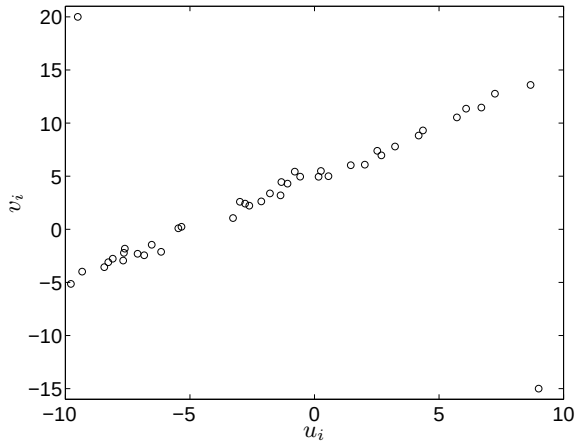
Newton's method

13.3 For the data and with the same notation as in exercise 8.6, compute the coordinates (u_1, v_1), (u_2, v_2), (u_3, v_3) that minimize

$$l_1^4 + l_2^4 + l_3^4 + l_5^4 + l_6^4 + l_7^4.$$

Use Newton's method, with $u = v = 0$ as starting point. With this starting point no line search should be necessary.

13.4 The figure shows 42 data points (u_i, v_i). The points are approximately on a straight line, except for the first and the last point.



The data are available in the file `ch13ex4.m` as `[u,v] = ch13ex4.m`.

(a) Fit a straight line $v = \alpha + \beta u$ to the points by solving a least-squares problem

$$\text{minimize} \quad g(\alpha, \beta) = \sum_{i=1}^{42} (\alpha + \beta u_i - v_i)^2,$$

with variables α, β.

(b) Fit a straight line $v = \alpha + \beta u$ to the points by solving the unconstrained minimization problem

$$\text{minimize} \quad g(\alpha, \beta) = \sum_{i=1}^{42} \sqrt{(\alpha + \beta u_i - v_i)^2 + 25}$$

using Newton's method. Use as initial points the values of α and β computed in part (a). (With this starting point, no line search should be necessary; for other starting points, a line search might be needed.) Terminate the iteration when $\|\nabla g(\alpha, \beta)\| \leq 10^{-6}$.

(c) Fit a straight line $v = \alpha + \beta u$ to the points by solving the unconstrained minimization problem

$$\text{minimize} \quad g(\alpha, \beta) = \sum_{i=1}^{42} (\alpha + \beta u_i - v_i)^4$$

using Newton's method. Use as initial points the values of α and β computed in part (a). (With this starting point, no line search should be necessary.) Terminate the iteration when $\|\nabla g(\alpha, \beta)\| \leq 10^{-6}$.

(d) Plot the three functions $\alpha + \beta u$ computed in parts (a), (b) and (c) versus u, and compare the results.

13.5 Consider the problem of minimizing the function

$$\begin{aligned} g(x) &= \sum_{i=1}^{n} \exp(x_i + a_i) + \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2 \\ &= \sum_{i=1}^{n} \exp(x_i + a_i) + (x_2 - x_1)^2 + (x_3 - x_2)^2 + \cdots + (x_n - x_{n-1})^2. \end{aligned}$$

The n-vector a is given.

(a) Give the gradient and Hessian of g. Show that the Hessian is positive definite everywhere.

(b) Describe an efficient method for computing the Newton step

$$v = -\nabla^2 g(x)^{-1} \nabla g(x).$$

What is the cost of your method (number of flops for large n)? It is sufficient to give the exponent of the dominant term in the flop count.

13.6 Define

$$g(x) = \|x - a\|^2 + \sum_{k=1}^{n-1} \sqrt{(x_{k+1} - x_k)^2 + \rho},$$

where a is a given n-vector and ρ is a given positive scalar. The variable is the n-vector x.

(a) Give expressions for the gradient and Hessian of g. Show that the Hessian is positive definite at all x.

(b) Describe an efficient method for computing the Newton step

$$v = -\nabla^2 g(x)^{-1} \nabla g(x).$$

Is the complexity of your method linear, quadratic, or cubic in n?

Chapter 14

Nonlinear least-squares

14.1 Definition

A nonlinear least-squares problem is an unconstrained minimization problem with objective function

$$g(x) = \sum_{i=1}^{m} r_i(x)^2, \tag{14.1}$$

where each $r_i(x)$ is a (possibly nonlinear) function of n variables. As a special case, if $r_i(x) = a_i^T x - b_i$, the problem reduces to a linear least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{m} (a_i^T x - b_i)^2.$$

Nonlinear least-squares problems, in which the functions r_i are not of the form $a_i^T x - b_i$, are much more difficult to solve than linear least-squares problems. The function g is usually not convex, and may have several local minima, so finding the global minimum can be very difficult. However, as we will see, there exist good methods for finding a local minimum.

Example We will use the following example to illustrate the ideas. We take $n = 2$, $m = 5$, and

$$r_i(x) = \sqrt{(x_1 - p_i)^2 + (x_2 - q_i)^2} - \rho_i. \tag{14.2}$$

The parameters p_i, q_i and ρ_i are given by

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.0 \\ 1.5 \\ 1.5 \\ 2.5 \end{bmatrix}, \qquad \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1.7 \\ 1.5 \\ 2.0 \\ 1.5 \end{bmatrix}, \qquad \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \\ \rho_5 \end{bmatrix} = \begin{bmatrix} 1.87 \\ 1.24 \\ 0.53 \\ 1.29 \\ 1.49 \end{bmatrix}.$$

The graph and the contour lines of g are shown in figure 14.1. The function is clearly not convex. Its minimum is at $(1.18, 0.82)$, but there is also a shallow local minimum at $(2.99, 2.12)$ and a local maximum at $(1.94, 1, 87)$.

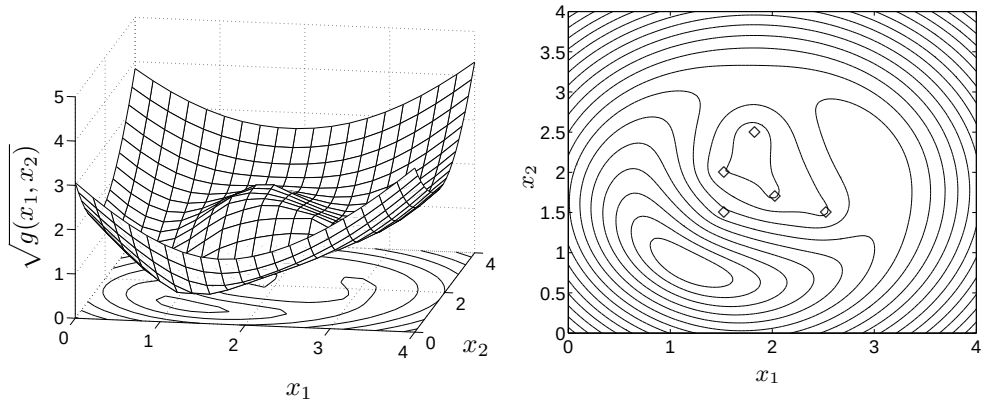


Figure 14.1 Graph and contour lines of the function $g = \sum_i r_i(x)^2$ with r_i defined in (14.2). The diamonds indicate the points (p_i, q_i). The minimum of g is $x = (1.18, 0.82)$. There is a local minimum at $x = (2.99, 2.12)$, and a local maximum at $(1.94, 1.87)$.

14.2 Newton's method

We can approach the nonlinear least-squares problem as a general unconstrained minimization problem, and apply Newton's method (see section 13.6).

The following formulas give the gradient and Hessian of g in terms of the gradients and Hessians of the functions r_i:

$$\nabla g(x) = 2\sum_{i=1}^{m} r_i(x)\nabla r_i(x) \tag{14.3}$$

$$\nabla^2 g(x) = 2\sum_{i=1}^{m} \left(r_i(x)\nabla^2 r_i(x) + \nabla r_i(x)\nabla r_i(x)^T\right). \tag{14.4}$$

Example The gradient and Hessian of the function r_i defined in (14.2) are

$$\begin{aligned}
\nabla r_i(x) &= (1/d_i)\begin{bmatrix} x_1 - p_i \\ x_2 - q_i \end{bmatrix} \\
\nabla^2 r_i(x) &= (1/d_i^3)\begin{bmatrix} d_i^2 - (x_1 - p_i)^2 & -(x_1 - p_i)(x_2 - q_i) \\ -(x_1 - p_i)(x_2 - q_i) & d_i^2 - (x_2 - q_i)^2 \end{bmatrix}
\end{aligned}$$

where we introduce $d_i = \sqrt{(x_1 - p_i)^2 + (x_2 - q_i)^2}$ to simplify the notation.

When started at $x^{(0)} = (1.5, 4)$, Newton's method produces the iterates shown in figure 14.2. The iterates first pass through a region in which $\nabla^2 g(x)$ is not positive definite, so the negative gradient direction is used. From the third iteration onwards, the Newton step was used.

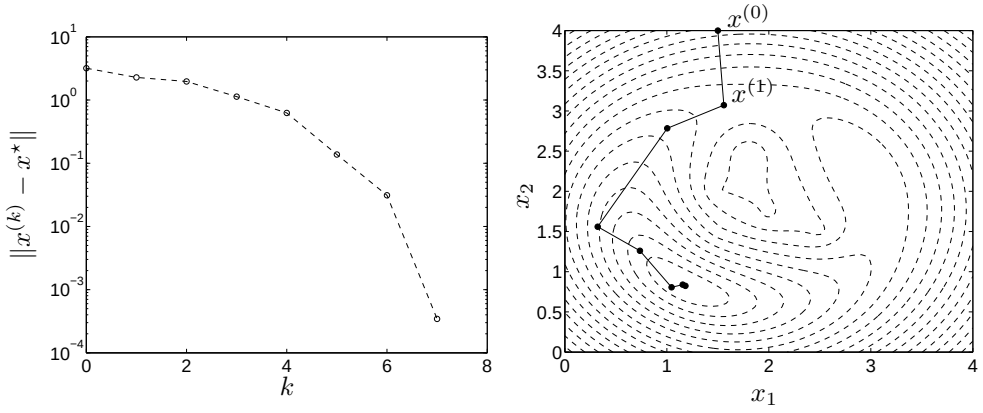


Figure 14.2 The iterates in Newton's method for local minimization, applied to the example problem of figure 14.1.

14.3 Gauss-Newton method

A simpler method for solving nonlinear least-squares problems is the *Gauss-Newton method*. The idea is as follows. We start at some initial guess $x^{(0)}$. At each iteration, we calculate the first-order approximations of the functions r_i around $x^{(k)}$:

$$r_i(x) \approx r_{\mathrm{aff},i}(x) = r_i(x^{(k)}) + \nabla r_i(x^{(k)})^T(x - x^{(k)}).$$

We then replace the functions r_i in the cost function g by their first-order approximations and solve the resulting minimization problem

$$\text{minimize} \quad \sum_{i=1}^{m} r_{\mathrm{aff},i}(x)^2. \tag{14.5}$$

The solution is taken as our next iterate $x^{(k+1)}$.

The minimization problem (14.5) is a *linear* least-squares problem, because the objective function

$$\sum_{i=1}^{m} r_{\mathrm{aff},i}(x)^2 = \sum_{i=1}^{m} \left(r_i(x^{(k)}) + \nabla r_i(x^{(k)})^T(x - x^{(k)}) \right)^2$$

can be expressed as

$$\sum_{i=1}^{m} r_{\mathrm{aff},i}(x)^2 = \|A^{(k)}x - b^{(k)}\|^2$$

with

$$A^{(k)} = \begin{bmatrix} \nabla r_1(x^{(k)})^T \\ \nabla r_2(x^{(k)})^T \\ \vdots \\ \nabla r_m(x^{(k)})^T \end{bmatrix}, \qquad b^{(k)} = \begin{bmatrix} \nabla r_1(x^{(k)})^T x^{(k)} - r_1(x^{(k)}) \\ \nabla r_2(x^{(k)})^T x^{(k)} - r_2(x^{(k)}) \\ \vdots \\ \nabla r_m(x^{(k)})^T x^{(k)} - r_m(x^{(k)}) \end{bmatrix}.$$

The Gauss-Newton method can be summarized as follows. We assume that $A^{(k)}$ is left-invertible.

Algorithm 14.1. GAUSS-NEWTON METHOD FOR NONLINEAR LEAST-SQUARES.

given initial x, tolerance $\epsilon > 0$.
repeat

1. Evaluate $r_i(x)$ and $\nabla r_i(x)$ for $i = 1, \ldots, m$, and calculate

$$r := \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_m(x) \end{bmatrix}, \qquad A := \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}, \qquad b := Ax - r.$$

2. **if** $\|2A^T r\| \leq \epsilon$, **return** x.
3. $x := (A^T A)^{-1} A^T b$.

until a limit on the number of iterations is exceeded

In step 2, we evaluate $g(x)$ as

$$\nabla g(x) = 2 \sum_{i=1}^{m} r_i(x) \nabla r_i(x) = 2A^T r,$$

The vector x in step 3 is the solution of a linear least-squares problem (minimize $\|Ax - b\|^2$), and can be computed using the QR factorization or the Cholesky factorization.

The Gauss-Newton method does not require the second derivatives $\nabla^2 r_i(x)$. Newton's method, on the other hand, requires the second derivatives to form $\nabla^2 g(x)$ (see equation (14.4)).

Interpretation as simplified Newton method The Gauss-Newton update

$$x^+ = (A^T A)^{-1} A^T b$$

in step 3 can be interpreted as an approximation of the Newton update

$$x^+ = x - \nabla^2 g(x)^{-1} \nabla g(x).$$

To see the connection, we first note that $b = Ax - r$, so

$$\begin{aligned} x^+ &= (A^T A)^{-1} A^T b \\ &= (A^T A)^{-1} A^T (Ax - r) \\ &= x - (A^T A)^{-1} A^T r. \end{aligned} \tag{14.6}$$

As we have seen, $\nabla g(x) = 2A^T r$, so x^+ can also be expressed as

$$x^+ = x - (2A^T A)^{-1} \nabla g(x).$$

This shows that x^+ is obtained from x by a 'Newton-like' update, with the true Newton step $v = -\nabla^2 g(x)^{-1}\nabla g(x)$ replaced with

$$v = -(2A^T A)^{-1}\nabla g(x). \tag{14.7}$$

The matrix $H = 2A^T A$ is actually closely related to $\nabla^2 g(x)$, as can be seen from

$$\begin{aligned} H = 2A^T A &= 2\begin{bmatrix} \nabla r_1(x) & \nabla r_2(x) & \cdots & \nabla r_m(x) \end{bmatrix} \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix} \\ &= 2\sum_{i=1}^m \nabla r_i(x)\nabla r_i(x)^T. \end{aligned}$$

Comparing with (14.4) we can interpret H as an approximate Hessian, obtained by omitting the terms $r_i(x)\nabla^2 r_i(x)$. We can expect the approximation to be accurate (and hence, the Gauss-Newton method to behave very much like Newton's method), in the neighborhood of a solution $x^\star$ where the values of $r_i(x^\star)$ are small.

Gauss-Newton method with backtracking The Gauss-Newton method converges if started near a local minimum. We can add a line search to improve its convergence range: at each iteration we compute the 'Gauss-Newton direction' v given by (14.7), determine a step size t by backtracking, and take $x + tv$ as next iterate.

As an interesting detail, it can be shown that the Gauss-Newton direction v is a descent direction, because $\nabla g(x)^T v = -2v^T(A^T A)v < 0$ because the matrix $A^T A$ is positive definite.

Algorithm 14.2. Gauss-Newton method with backtracking.

given initial x, tolerance $\epsilon > 0$, parameter $\alpha \in (0, 1/2)$
repeat

1. Evaluate $r_i(x)$ and $\nabla r_i(x)$ for $i = 1, \ldots, m$, and calculate
$$r := \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_m(x) \end{bmatrix}, \qquad A := \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}.$$
2. **if** $\|2A^T r\| \leq \epsilon$, **return** x.
3. $v := -(A^T A)^{-1}A^T r$.
4. $t := 1$
while $\sum_{i=1}^m r_i(x + tv)^2 > \|r\|^2 + \alpha(2r^T Av)t$,
$t := t/2$.
5. $x := x + tv$.

until a limit on the number of iterations is exceeded

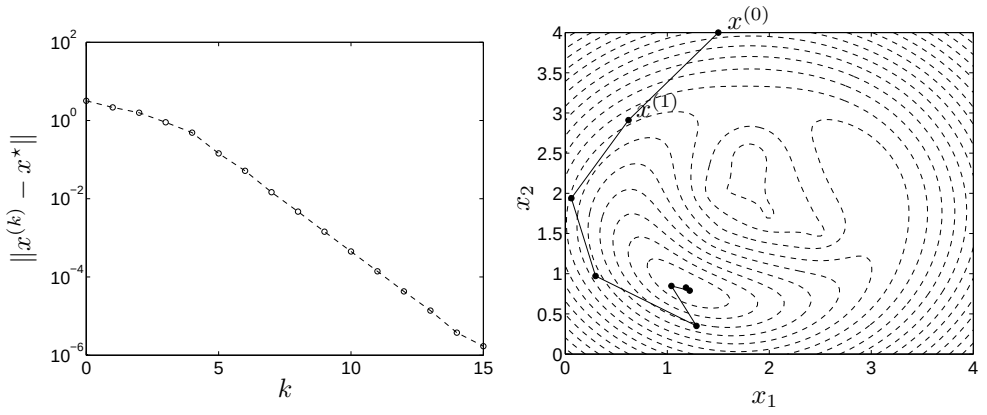


Figure 14.3 The iterates in Gauss-Newton method applied to the example problem of figure 14.1.

In step 3, we use the expression for the Gauss-Newton step given in (14.6). The update v can be computed by solving a linear least-squares problem (minimize $\|Av + r\|^2$).

Two other simplifications were made in the evaluation of the backtracking condition in step 4. First, we calculate $g(x) = \sum_{i=1}^m r_i(x)^2$ as $g(x) = \|r\|^2$. Second, the coefficient of t follows from the fact that $\nabla g(x) = 2A^T r$, so

$$\nabla g(x)^T v = 2r^T Av.$$

Example The code for the example problem of figure 14.2 (with starting point $x^{(0)} = (1.5, 4)$, and $\alpha = 0.01$) is as follows. The result is shown in figure 14.3.

```
m = 5;  n = 2;
p = [1.8;  2.0;  1.5;  1.5;  2.5];
q = [2.5;  1.7;  1.5;  2.0;  1.5];
rho = [1.87;  1.24;  0.53;  1.29;  1.49];
x = [1.5; 4];
for k=1:50
   d = sqrt((x(1)-p).^2 + (x(2)-q).^2);
   r = d - rho;
   A = [(x(1)-p)./d  (x(2)-q)./d];
   if (norm(2*A'*r) < 1e-5) break; end;
   v = -A\r;
   t = 1;
   newx = x+t*v;
   newr = sqrt((newx(1)-p).^2 + (newx(2)-q).^2) - rho;
   while (norm(newr)^2 > norm(r)^2 + 0.01*(2*r'*A*v)*t)
      t = t/2;
      newx = x+t*v;
```

```
        newr = sqrt((newx(1)-p).^2 + (newx(2)-q).^2) - rho;
      end;
      x = x + t*v;
    end;
```

Exercises

14.1 The federal government has mandated that cellular network operators must have the ability to locate a cell phone from which an emergency call is made. This problem concerns a simplified problem in which we can use time of arrival information at a number of base stations to estimate the cell phone location.

A cell phone at location (x_1, x_2) in a plane (we assume that the elevation is zero for simplicity) transmits an emergency signal at time x_3. This signal is received at m base stations, located at positions (p_1, q_1), (p_2, q_2), ..., (p_m, q_m). Each base station can measure the time of arrival of the emergency signal, within a few tens of nanoseconds. The measured times of arrival are

$$\tau_i = \frac{1}{c}\sqrt{(x_1 - p_i)^2 + (x_2 - q_i)^2} + x_3 + v_i, \quad i = 1, \ldots, m,$$

where c is the speed of light (0.3 meters/nanosecond), and v_i is the noise or error in the measured time of arrival. The problem is to estimate the cell phone position $x = (x_1, x_2)$, as well as the time of transmission x_3, based on the time of arrival measurements $\tau_1, \ldots, \tau_m$.

The mfile `ch14ex1.m`, available on the course web site, defines the data for this problem. It is executed as `[p,q,tau] = ch14ex1`. The 9×1-arrays `p` and `q` give the positions of the 9 base stations. The 9×1-array `tau` contains the measured times of arrival. Distances are given in meters and times in nanoseconds.

Determine an estimate $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$ of the unknown coordinates and the time of transmission, by solving the nonlinear least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{9} r_i(x)^2$$

where

$$r_i(x_1, x_2, x_3) = \frac{1}{c}\sqrt{(x_1 - p_i)^2 + (x_2 - q_i)^2} + x_3 - \tau_i.$$

Use the Gauss-Newton method, with $x_1 = x_2 = x_3 = 0$ as starting point. From this starting point, the method should converge without backtracking, and you do not have to include backtracking in your code.

Your solution should include:

- a description of the least-squares problems that you solve at each iteration
- the computed estimates $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$
- a plot of $g(x^{(k)}) - g(\hat{x})$ versus k, where $g(x) = \sum_{i=1}^{9} r_i(x)^2$.

14.2 In exercise 8.4 we developed a simple approximate model for the inductance of a planar CMOS inductor. The model had the form

$$\hat{L} = \alpha n^{\beta_1} w^{\beta_2} d^{\beta_3} D^{\beta_4},$$

and depended on five model parameters: α, β_1, β_2, β_3, β_4. We calculated the five parameters by minimizing the error function

$$\sum_{i=1}^{50} (\log L_i - \log \hat{L}_i)^2,$$

which is a linear least-squares problem in the variables

$$x_1 = \log \alpha, \qquad x_2 = \beta_1, \qquad x_3 = \beta_2, \qquad x_4 = \beta_3, \qquad x_5 = \beta_4.$$

In this problem you are asked to estimate the model parameters by minimizing the function

$$g(x) = \sum_{i=1}^{50} (L_i - \hat{L}_i)^2.$$

This is a nonlinear least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{50} r_i(x)^2$$

with x defined as before, and

$$r_i(x) = e^{x_1} n_i^{x_2} w_i^{x_3} d_i^{x_4} D_i^{x_5} - L_i.$$

Solve this nonlinear least-squares problem using the Gauss-Newton method with backtracking line search, and the data from `ch8ex4.m`.

Consider two starting points: (1) the answer of exercise 8.4, *i.e.*, the values of the model parameters calculated by solving the linear least-squares problem; (2) $x = 0$. For both starting points, plot the step sizes $t^{(k)}$ and the error $g(x^{(k)}) - g(x^\star)$ versus the iteration number k.

14.3 Suppose you are asked to fit the following three functions $f(v, w)$ to experimental data:

(a) $f(v, w) = \alpha v^\beta w^\gamma (v + w)^\delta$

(b) $f(v, w) = \alpha v^\beta + \gamma w^\delta$

(c) $f(v, w) = \alpha v^\beta w^\gamma / (1 + \alpha v^\beta w^\gamma)$.

Each function has two variables (v, w), and depends on several parameters (α, β, γ, δ). Your task is to calculate values of these parameters such that

$$f(v_i, w_i) \approx t_i, \quad i = 1, \ldots, N.$$

The problem data t_i, v_i, w_i are given. We assume that $N > 4$, and $v_i > 0$, $w_i > 0$, $t_i > 0$. In the third subproblem, we also assume that $t_i < 1$. The exponents of v, w, and $v + w$ in the definitions of f are allowed to be positive, negative, or zero, and do not have to be integers.

For each of the three functions, explain how you would formulate the model fitting problem as a linear least-squares problem

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

or as a nonlinear least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{m} r_i(x)^2.$$

You are free to use any reasonable error criterion to judge the quality of the fit between the experimental data and the model. Your solution must include a clear statement of the following:

- the variables x in the linear or nonlinear least-squares problem
- the error criterion that you minimize
- if you use linear least-squares: the matrix A and the vector b
- if you use nonlinear least-squares: the functions $r_i(x)$, and the matrix $A^{(k)}$ and the vector $b^{(k)}$ in the linear least-squares problem

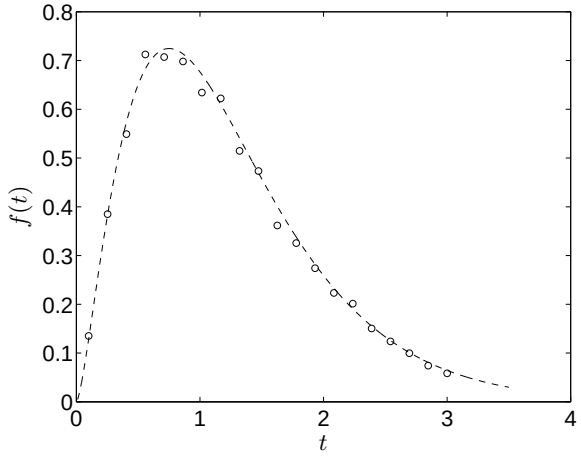
  $$\text{minimize} \quad \|A^{(k)}x - b^{(k)}\|^2$$

  that you solve at iteration k of the Gauss-Newton method.

14.4 The figure shows $m = 20$ points (t_i, y_i) as circles. These points are well approximated by a function of the form

$$f(t) = \alpha t^{\beta} e^{\gamma t}.$$

(An example is shown in dashed line.)



Explain how you would compute values of the parameters α, β, γ such that

$$\alpha t_i^{\beta} e^{\gamma t_i} \approx y_i, \quad i = 1, \ldots, m, \tag{14.8}$$

using the following two methods.

(a) The Gauss-Newton method applied to the nonlinear least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{m} \left(\alpha t_i^{\beta} e^{\gamma t_i} - y_i\right)^2$$

with variables α, β, γ. Your description should include a clear statement of the linear least-squares problems you solve at each iteration. You do not have to include a line search.

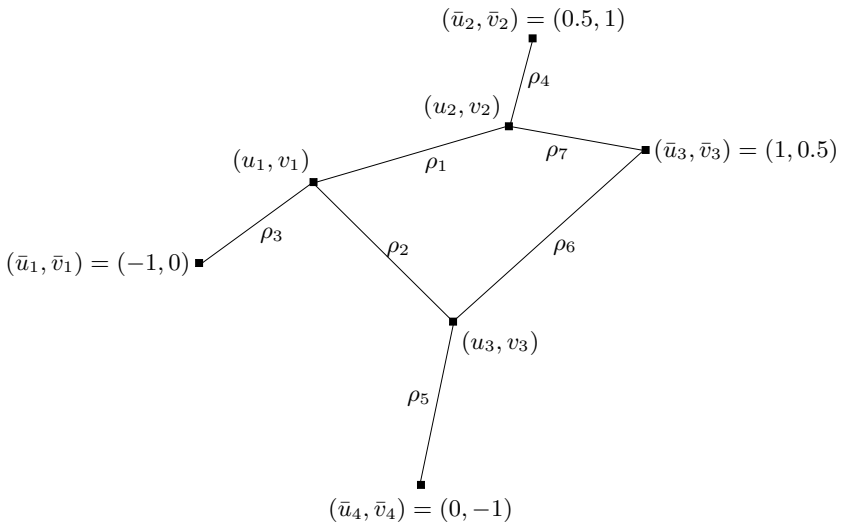
(b) Solving a single linear least-squares problem, obtained by selecting a suitable error function for (14.8) and/or making a change of variables. Clearly state the least-squares problem, the relation between its variables and the parameters α, β, γ, and the error function you choose to measure the quality of fit in (14.8).

14.5 In this exercise we use the Gauss-Newton method to solve the following problem: determine the coordinates of N points in a plane, given the coordinates of M other points with known positions, and (noisy) measurements of the distances between certain pairs of the $M + N$ points. We call the N points with unknown coordinates the *free points*, and the M points with known coordinates the *anchor points*.

In a practical application, the points correspond to nodes in a wireless sensor network. Some nodes (the anchor nodes) have been carefully placed or are equipped with GPS receivers, so their coordinates are known. The coordinates of the other nodes (the free nodes) have to be determined from measurements of the distances to neighboring nodes.

The problem can be represented as a graph. The $M+N$ nodes of the graph represent the N free points and the M anchor points. The coordinates of the free nodes are denoted (u_1, v_1), $\ldots$, (u_N, v_N). They are the variables in the problem. The coordinates of the anchor points are denoted $(\bar{u}_1, \bar{v}_1)$, $\ldots$, $(\bar{u}_M, \bar{v}_M)$ and are given. The K edges in the graph represent the measurements: if there is an edge between two nodes in the graph, a measurement is made of the distance between the corresponding points.

An example with 4 anchor points, 3 free points, and 7 distance measurements is shown below.



We will formulate the problem in terms of a $K \times N$-matrix B and three K-vectors c, d, ρ, defined as follows. The rows in B and the elements of b, c, and ρ correspond to the different edges in the graph.

- If row k corresponds to an edge between free points i and j, we take

  $$b_{ki} = 1, \qquad b_{kj} = -1, \qquad c_k = d_k = 0.$$

  The other elements of the kth row of B are zero. ρ_k is a (noisy) measurement of the distance $((u_i - u_j)^2 + (v_i - v_j)^2)^{1/2}$.

- If row k corresponds to an edge between free point i and anchor point j, we take

  $$b_{ki} = 1, \qquad c_k = -\hat{u}_j, \qquad d_k = -\hat{v}_j.$$

  The other elements in row k of B are zero. ρ_k is a measurement of the distance $((u_i - \hat{u}_j)^2 + (v_i - \hat{v}_j)^2)^{1/2}$.

With this notation, the length of edge k is

$$l_k(u, v) = \sqrt{(b_k^T u + c_k)^2 + (b_k^T v + d_k)^2}$$

where b_k^T is the kth row of B and $u = (u_1, \ldots, u_N)$, $v = (v_1, \ldots, v_N)$.
For the example of the figure, we can define B, c and d as

$$B = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad c = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -0.5 \\ 0 \\ -1 \\ -1 \end{bmatrix}, \qquad d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ -0.5 \\ -0.5 \end{bmatrix}.$$

The problem is to find values of u, v that satisfy $l_k(u,v) \approx \rho_k$ for $k = 1, \ldots, K$. This can be posed as a nonlinear least-squares problem

$$g(u,v) = \sum_{k=1}^{K} r_k(u,v)^2 \tag{14.9}$$

with variables u and v, where

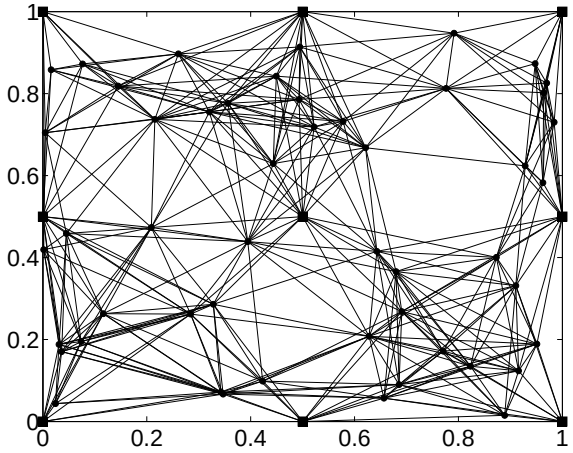
$$\begin{aligned} r_k(u,v) &= l_k(u,v)^2 - \rho_k^2 \\ &= (b_k^T u + c_k)^2 + (b_k^T v + d_k)^2 - \rho_k^2. \end{aligned}$$

(Here we define r_k as $l_k(u,v)^2 - \rho_k^2$ rather than $l_k(u,v) - \rho_k$ to simplify the calculation of the derivates.)

The file `ch14ex5.m` on the class webpage contains the data that we will use in the problem. It can be executed in MATLAB using the command

```
[B, c, d, rho] = ch14ex5;
```

This creates the problem data B, c, d, ρ for the network shown in the figure below.



There are 50 free nodes ($N = 50$), 9 anchor nodes shown as squares ($M = 9$), and 389 edges ($K = 389$).

Find estimates of u and v by solving (14.9) using the Gauss-Newton method. When testing your code, try several randomly generated starting points in the square $[0,1] \times [0,1]$ (using the MATLAB commands `u = rand(N,1)`, `v = rand(N,1)`). Terminate the iteration when $\|\nabla g(u,v)\|_2 \leq 10^{-6}$. You can follow the outline of Algorithm 14.2 in the course reader (Gauss-Newton algorithm with backtracking), with $\alpha = 0.01$ and $\epsilon = 10^{-6}$. However for this problem the Gauss-Newton method converges without backtracking for most starting points (in fact, for all the starting points that we tried), so it is not necessary to include a backtracking line search, and you can also follow Algorithm 14.1.

14.6 As in exercise 8.11 we consider the problem of fitting a circle to m points in a plane, but with a different error function. In this exercise we formulate the problem as a nonlinear least-squares problem

$$\text{minimize} \quad g(u_c, v_c, R) = \sum_{i=1}^{m} \left(\sqrt{(u_i - u_c)^2 + (v_i - v_c)^2} - R \right)^2$$

with variables u_c, v_c, R. Note that $|\sqrt{(u_i - u_c)^2 + (v_i - v_c)^2} - R|$ is the distance of the point (u_i, v_i) to the circle, so in this formulation we minimize the sum of the squared distances of the points to the circle.

Apply the Gauss-Newton method to this nonlinear least-squars problem with the data in `ch8ex11.m`. Terminate the iteration when $\|\nabla g(u_c, v_c, R)\| \leq 10^{-5}$. You can select a good starting point from the plot of the 50 points (u_i, v_i) or from the solution of exercise 8.11. With a good starting point, no backtracking line search will be needed, so you can follow the outline of Algorithm 14.1.

Include in your solution:

- a description of the least-squares problem you solve at each iteration (the matrix A and the vector b)
- the MATLAB code
- the computed solution u_c, v_c, R
- a plot of the computed circle, created with the commands

  ```
  t = linspace(0, 2*pi, 1000);
  plot(u, v, 'o',  R * cos(t) + uc, R * sin(t) + vc, '-');
  axis square
  ```

  (assuming your MATLAB variables are called `uc`, `vc`, and `R`).

Chapter 15

Linear optimization

15.1 Definition

A *linear program* (LP) is an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{subject to} & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \\ & \cdots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m. \end{array}$$

The problem has n optimization or decision variables x_1, x_2, ..., x_n, and a linear cost (or objective) function $c_1x_1 + \cdots + c_nx_n$. The inequalities are called the *constraints* of the problem. A vector $x = (x_1, \ldots, x_n)$ is *feasible* if it satisfies all the constraints. A feasible point is a *solution* of the LP if there is no feasible point with a lower value of the cost function. If x is a solution, then $c_1x_1 + \cdots + c_nx_n$ is called the *optimal value* of the LP.

It is convenient to write the problem using matrix-vector notation as

$$\begin{array}{ll} \text{minimize} & c^Tx \\ \text{subject to} & Ax \leq b, \end{array} \tag{15.1}$$

where A is the $m \times n$-matrix with entries a_{ij}. The inequality $Ax \leq b$ is interpreted elementwise, *i.e.*, $Ax \leq b$ is equivalent to $(Ax)_i \leq b_i$ for $i = 1, \ldots, m$ (see section 1.9).

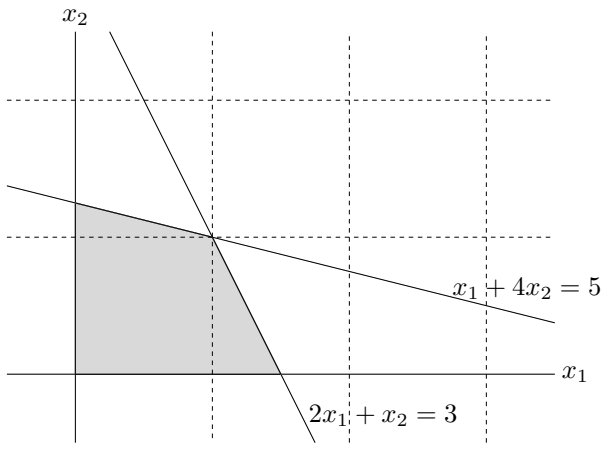


Figure 15.1 Feasible set of the LP (15.2).

15.2 Examples

We begin with a two-variable LP that will illustrate some important aspects of the general problem:

$$\begin{array}{ll} \text{minimize} & -x_1 - x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 3 \\ & x_1 + 4x_2 \leq 5 \\ & x_1 \geq 0, \quad x_2 \geq 0. \end{array} \tag{15.2}$$

This LP can be written in the matrix-vector format (15.1) by defining

$$c = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \qquad A = \begin{bmatrix} 2 & 1 \\ 1 & 4 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} 3 \\ 5 \\ 0 \\ 0 \end{bmatrix}.$$

The feasible points are shown as the shaded region in figure 15.1. This set is called the *feasible set* of the LP.

Figure 15.2 shows a graphical construction of the solution. The dashed lines are the contour lines of the cost function $-x_1 - x_2$. The solution of the LP is the feasible point of minimal cost, *i.e.*, the point $(1, 1)$. We note that the solution lies at one of the 'corners' of the feasible set. These corners are called the *extreme points* or *vertices*. We also note that the solution of this LP is unique, because all other feasible points have a cost function strictly greater than -2.

In the next example we keep the constraints of the first problem but change the cost function to $-2x_1 - x_2$:

$$\begin{array}{ll} \text{minimize} & -2x_1 - x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 3 \\ & x_1 + 4x_2 \leq 5 \\ & x_1 \geq 0, \quad x_2 \geq 0. \end{array} \tag{15.3}$$

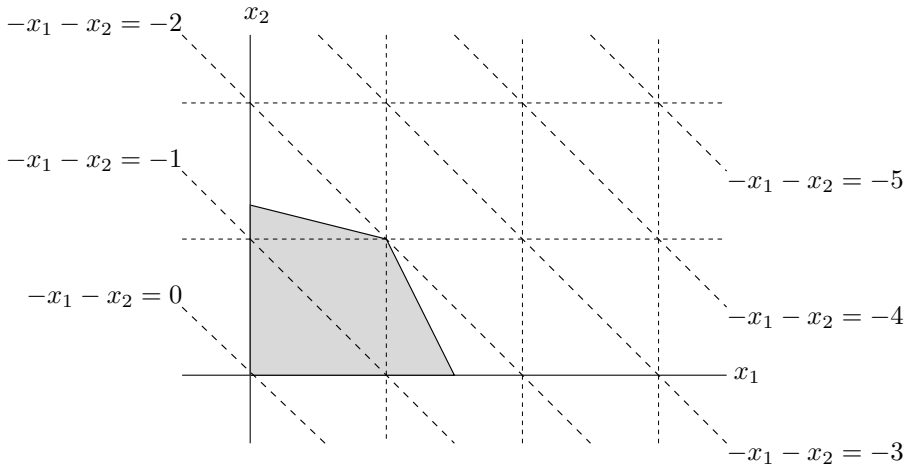


Figure 15.2 Lines of constant cost of the LP (15.2). The solution is $x = (1, 1)$.

If we apply the same graphical construction (figure 15.3), we find that the optimal solution is not unique. The entire line segment between the extreme points $(1, 1)$ and $(1.5, 0)$ is optimal.

Another possibility is illustrated by the LP

$$\begin{array}{ll} \text{minimize} & -x_1 - x_2 \\ \text{subject to} & -2x_1 - 3x_2 \leq -3 \\ & 2x_1 - 3x_2 \leq 3 \\ & x_1 \geq 0. \end{array} \tag{15.4}$$

(See figure 15.4.) This LP does not have a solution because we can find feasible points with arbitrarily small cost value. We say the problem is *unbounded below* and that the optimal value is $-\infty$.

Adding the constraint $x_1 + x_2 \leq -1$ to the LP (15.2) gives

$$\begin{array}{ll} \text{minimize} & -x_1 - x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 3 \\ & x_1 + 4x_2 \leq 5 \\ & x_1 + x_2 \leq -1 \\ & x_1 \geq 0, \quad x_2 \geq 0. \end{array}$$

The feasible set is empty: it is the set shown in figure 15.1 with the extra requirement that $x_1 + x_2 \leq -1$. This is an example of an *infeasible* LP, a problem with no feasible points. An infeasible LP has no solution and we define its optimal value to be $+\infty$.

The examples give LPs of the following three categories.

- *Optimal value* $+\infty$. The LP is infeasible, *i.e.*, the constraints are inconsistent.
- *Finite optimal value.* The LP has at least one solution. If there are multiple solutions, they all have the same objective value.

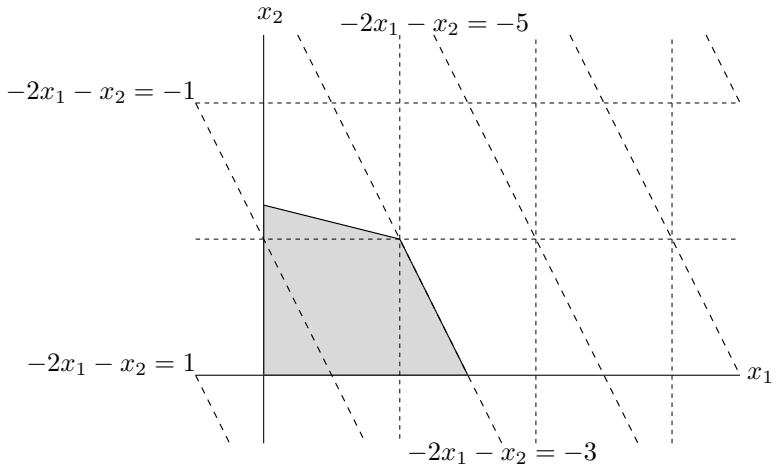


Figure 15.3 Lines of constant cost of the LP (15.3). Every point on the line segment between $x = (1, 1)$ and $x = (1.5, 0)$ is a solution.

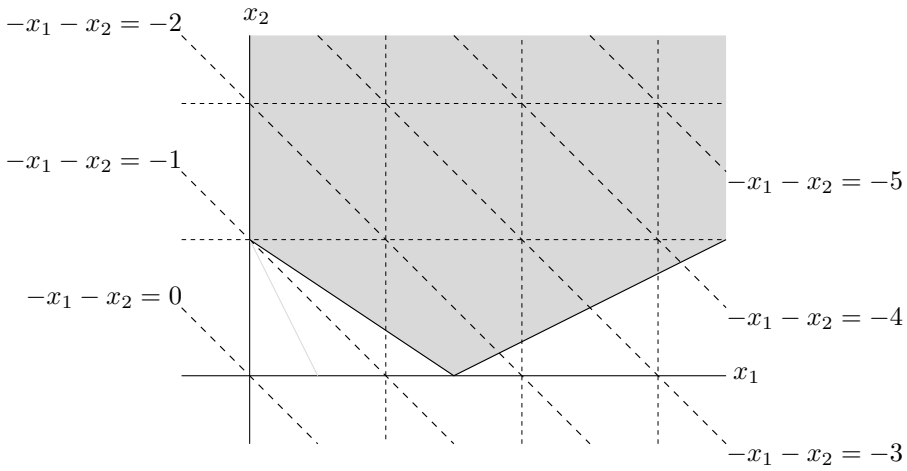


Figure 15.4 Feasible set and contour lines the LP (15.4). The problem is unbounded below.

- *Optimal value* $-\infty$. The LP is unbounded below, *i.e.*, we can find feasible points with arbitrarily small values of the cost function.

In the second case, the examples suggest that if an LP has a finite optimal value, then there is at least one solution at an extreme point. This is usually true, but there are exceptions. The LP

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 \\ \text{subject to} & -1 \leq x_1 + x_2 \leq 1, \end{array}$$

for example, has optimal value -1 and infinitely many solutions (any x with $x_1 + x_2 = -1$ is optimal). However the feasible set has no extreme points. This situation is rare in practice and can be excluded by requiring that A has a zero nullspace. It can be shown that if the feasible set of an LP has at least one extreme point and the optimal value is finite, then the LP has an optimal extreme point. (This will follow from the simplex algorithm in section 15.5.)

15.3 Polyhedra

In this section we examine the geometry of the solution of a set of linear inequalities.

Hyperplanes We first review the geometric interpretation of a single linear equation in n variables

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b.$$

When all the coefficients a_i are zero, the equation is trivial: the solution is either empty (if $b \neq 0$) or the entire space $\mathbf{R}^n$ (if $b = 0$). If $a = (a_1, a_2, \ldots, a_n) \neq 0$, the solution set

$$\{x \in \mathbf{R}^n \mid a^Tx = b\}$$

is called a *hyperplane*. Figure 15.5 illustrates the geometric interpretation of a hyperplane. The point

$$u = \frac{b}{\|a\|^2}a$$

is the multiple of a that lies in the hyperplane. Since $a^Tu = b$ the equation $a^Tx = b$ can be written as

$$a^T(x - u) = 0.$$

This shows that x is in the hyperplane if and only if the vector $x-u$ is orthogonal to a. For this reason, a is called the *normal vector* of the hyperplane. The hyperplane consists of an offset u, plus all vectors orthogonal to the normal vector a.

Note that the parameters a, b of a hyperplane are not unique. If $t \neq 0$, then the equation $(ta)^Tx = tb$ defines the same hyperplane as $a^Tx = b$.

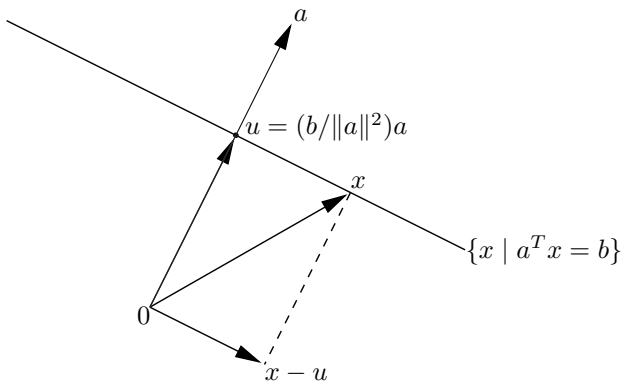


Figure 15.5 Hyperplane in $\mathbf{R}^2$ with normal vector a. A point x is in the hyperplane if $x - u$ is orthogonal to a.

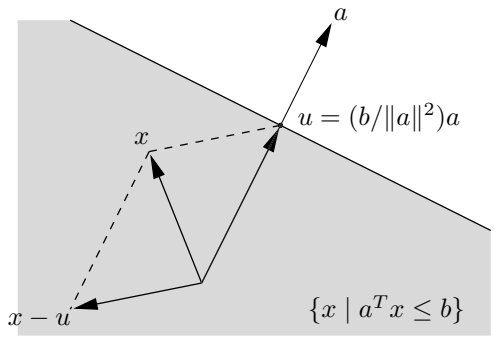


Figure 15.6 The shaded region (including the heavy line) is the halfspace defined by the inequality $a^T x \leq b$. A point x is in the halfspace if the vector $x - u$ makes an obtuse or right angle with the normal vector a.

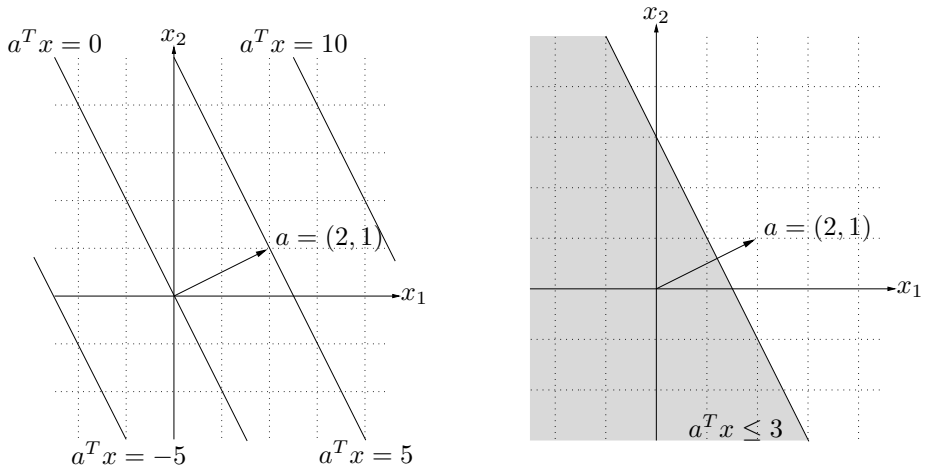


Figure 15.7 The figure on the left shows the hyperplanes $\{(x_1, x_2) \mid 2x_1 + x_2 = b\}$ for $b = -5$, 0, 5, 10. The shaded region on the right is the halfspace $\{(x_1, x_2) \mid 2x_1 + x_2 \leq 3\}$.

Halfspaces The hyperplane $\{x \mid a^T x = b\}$ divides $\mathbf{R}^n$ in two *halfspaces*

$$\{x \mid a^T x \leq b\}, \qquad \{x \mid a^T x \geq b\}.$$

Figure 15.6 illustrates the halfspace defined by the inequality $a^T x \leq b$. A point x is in this halfspace if $a^T x \leq b$ or $a^T(x - u) \leq 0$. In other words the angle between a and $x - u$ is greater than or equal to 90°. A point x is in the opposing halfspace defined by $a^T x \geq b$ if $a^T(x - u) \geq 0$, *i.e.*, the angle between a and $x - u$ is less than or equal to 90°.

The parameters a, b of a halfspace are not unique: if $t > 0$, then $(ta)^T x \leq tb$ defines the same halfspace as $a^T x \leq b$.

Example Figure 15.7 shows several hyperplanes and a halfspace with normal vector $a = (2, 1)$. The hyperplanes defined by equalities $2x_1 + x_2 = b$ are straight lines perpendicular to the normal vector $a = (2, 1)$. To determine the offset of each hyperplane it is sufficient to find one point on it, for example, the point $u = (b/\|a\|^2)a$, or the intersection with one of the coordinate axes. To find the position of the hyperplane for $b = 10$, for example, we can use the point $u = (b/\|a\|^2)a = (4, 2)$, or $x = (5, 0)$, $x = (0, 10)$. To find the halfspace defined by $2x_1 + x_2 \leq 3$, we first determine the hyperplane that bounds the halfspace. The correct halfspace follows from the property that a is the outward pointing normal to the halfspace.

Polyhedra A *polyhedron* is the intersection of finitely many halfspaces. Algebraically, it is any set that can be expressed as the solution set of a finite number of linear inequalities

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \quad \leq \quad b_1$$

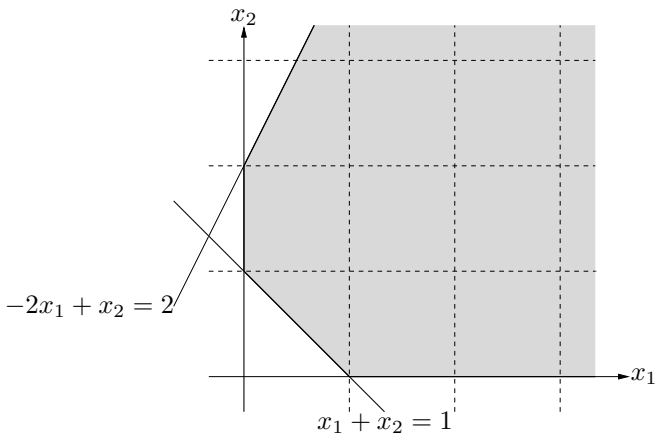


Figure 15.8 A polyhedron in $\mathbf{R}^2$.

$$\begin{array}{rcl} a_{21}x_1 + a_{22}x_2 + \cdots + a_{1n}x_n & \leq & b_2 \\ & \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \leq & b_m. \end{array}$$

Examples Figure 15.8 shows the polyhedron in $\mathbf{R}^2$ defined by the four inequalities

$$x_1 + x_2 \geq 1, \qquad -2x_1 + x_2 \leq 2, \qquad x_1 \geq 0, \qquad x_2 \geq 0. \tag{15.5}$$

In matrix-vector notation these inequalities are

$$\begin{bmatrix} -1 & -1 \\ -2 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} -1 \\ 2 \\ 0 \\ 0 \end{bmatrix}. \tag{15.6}$$

The inequality description of a polyhedron is by no means unique. For example, adding the inequality $x_1 \geq -1$ to (15.5) does not change the polyhedron. (The inequality $x_1 \geq -1$ is said to be *redundant*.) We also note that the polyhedron in figure 15.8 is unbounded.

Figure 15.9 shows the polyhedron in $\mathbf{R}^3$ defined by the seven inequalities

$$0 \leq x_1 \leq 2, \qquad 0 \leq x_2 \leq 2, \qquad 0 \leq x_3 \leq 2, \qquad x_1 + x_2 + x_3 \leq 5.$$

In matrix-vector form,

$$\begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 5 \end{bmatrix}. \tag{15.7}$$

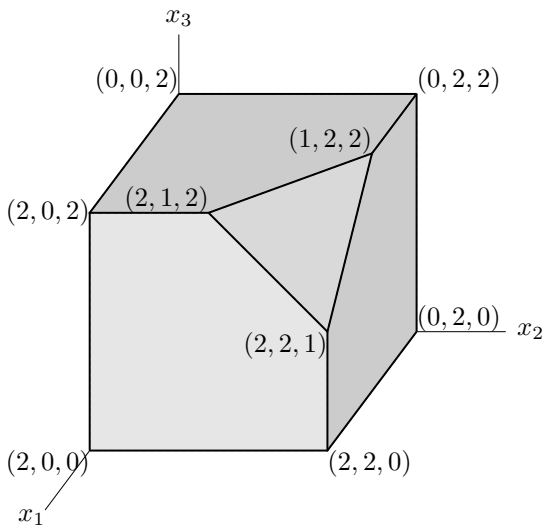


Figure 15.9 A polyhedron in $\mathbf{R}^3$ defined by the seven inequalities $0 \leq x_1 \leq 2$, $0 \leq x_2 \leq 2$, $0 \leq x_3 \leq 2$, $x_1 + x_2 + x_3 \leq 5$.

Another example is shown in figure 15.10. The shaded region is defined as

$$\{(x_1, x_2, x_3) \mid x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_1 + x_2 + x_3 = 1\}.$$

This is a polyhedron because it can be written as the solution set of five linear inequalities

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}. \tag{15.8}$$

15.4 Extreme points

The extreme points of a polyhedron play a fundamental role in the theory of linear programming. In this section we give a formal definition of the extreme points of a polyhedron

$$\mathcal{P} = \{x \in \mathbf{R}^n \mid Ax \leq b\}$$

defined by m inequalities in n variables: The rows of A will be denoted a_k^T, so the vector inequality $Ax \leq b$ is equivalent to m scalar inequalities

$$a_1^T x \leq b_1, \qquad a_2^T x \leq b_2, \qquad \ldots, \qquad a_m^T x \leq b_m.$$

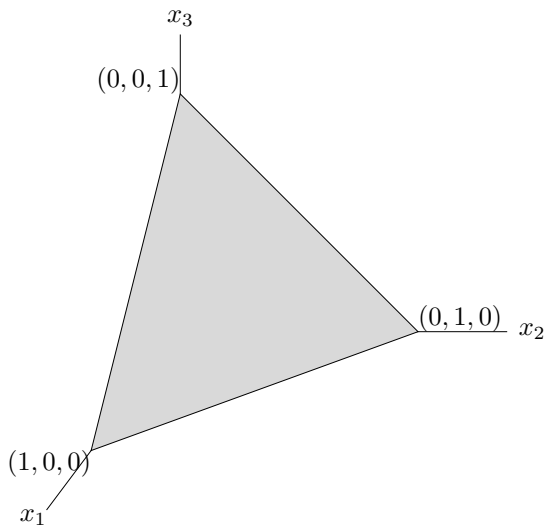


Figure 15.10 The polyhedron in $\mathbf{R}^3$ defined by the inequalities $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$ and the equality $x_1 + x_2 + x_3 = 1$.

Geometric definition A point $x \in \mathcal{P}$ is called an *extreme point* of $\mathcal{P}$ if it cannot be written as $x = (1/2)(y + z)$ where $y, z \in \mathcal{P}$ and $y \neq x$, $z \neq x$. In other words, x is an extreme point if it is not the midpoint between two other points in $\mathcal{P}$. This definition is illustrated in figure 15.11. The point $x^{(1)}$ in the figure is an extreme point because any line segment that has $x^{(1)}$ as its midpoint must have at least one endpoint outside $\mathcal{P}$. The line segment with end points $y^{(1)}$ and $z^{(1)}$, for example, has $x^{(1)}$ as its midpoint, but the endpoint $y^{(1)}$ is outside $\mathcal{P}$. The point $x^{(2)}$ is not an extreme point because it is the midpoint of a line segment with endpoints $y^{(2)}$ and $z^{(2)}$ in $\mathcal{P}$. Likewise the point $x^{(3)}$ is not an extreme point.

Algebraic definition The geometric definition in the previous paragraph coincides with the intuitive notion of a 'corner point' of a two- or three-dimensional polyhedron. However it is not clear how it can be applied in higher dimensions when it is not possible to visualize the polyhedron. In this section we give a useful alternative characterization, and later we will prove that the two definitions are equivalent.

Suppose x is feasible ($a_k^T x \leq b_k$ for $k = 1, \ldots, m$). We can partition the m inequalities in two groups. The inequalities that hold with equality ($a_k^T x = b_k$) are called the *active* inequalities at x. The other constraints ($a_k^T x < b_k$) are *inactive* at x. We will use the notation $I(x) \subseteq \{1, 2, \ldots, m\}$ to denote the set of indices of the active inequalities at x. With this notation

$$a_k^T x = b_k \text{ for } k \in I(x), \qquad a_k^T x < b_k \text{ for } k \notin I(x).$$

We can associate with the index set $I(x)$ a submatrix $A_{I(x)}$ that contains only the

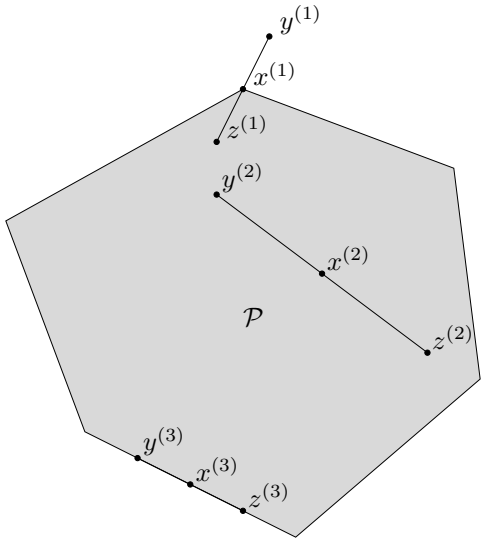


Figure 15.11 $x^{(1)}$ is an extreme point because all line segments with $x^{(1)}$ as midpoint have at least one endpoint outside $\mathcal{P}$. $x^{(2)}$ and $x^{(3)}$ are not extreme points because they are midpoints of line segments with both endpoints in $\mathcal{P}$.

rows of A indexed by $I(x)$. If $I(x) = \{k_1, k_2, \ldots, k_p\}$, then

$$A_{I(x)} = \begin{bmatrix} a_{k_1}^T \\ a_{k_2}^T \\ \vdots \\ a_{k_p}^T \end{bmatrix}.$$

The row dimension p of this matrix is the number of active constraints at x.

We will show that x is an extreme point if and only if the matrix $A_{I(x)}$ is left-invertible (*i.e.*, has a zero nullspace). Equivalently, we can find n linearly independent vectors among the normal vectors of the active constraints.

Examples The polyhedron in figure 15.8 is defined by the inequalities (15.6). It has three extreme points: $(1, 0)$, $(0, 1)$, $(0, 2)$. The active inequalities at $\hat{x} = (1, 0)$ are $x_1 + x_2 \geq 1$ and $x_2 \geq 0$. Therefore $I(\hat{x}) = \{1, 4\}$ and

$$A_{I(\hat{x})} = \begin{bmatrix} -1 & -1 \\ 0 & -1 \end{bmatrix}.$$

This matrix is square and nonsingular; therefore $\hat{x}$ is an extreme point. Similarly, the point $(0, 1)$ and $(0, 2)$ are extreme points because the submatrices

$$\begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix}, \qquad \begin{bmatrix} -2 & 1 \\ -1 & 0 \end{bmatrix}$$

are invertible. The point $\hat{x} = (2, 0)$ is not an extreme point. Only the inequality $-x_2 \leq 0$ is active at this point, so $I(\hat{x}) = \{4\}$ and

$$A_{I(\hat{x})} = \begin{bmatrix} 0 & -1 \end{bmatrix},$$

which is not left-invertible.

Next, we consider the three-dimensional polyhedron in figure 15.9 and the corresponding set of linear inequalities (15.7). The active inequalities at $\hat{x} = (2, 2, 1)$ are

$$x_1 \leq 2, \qquad x_2 \leq 2, \qquad x_1 + x_2 + x_3 \leq 5.$$

Therefore $I(\hat{x}) = \{2, 4, 7\}$ and

$$A_{I(\hat{x})} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

This matrix is square and nonsingular and therefore $\hat{x}$ is an extreme point.

Note that in figure 15.8 the extreme points are the feasible points at which exactly two inequalities are active. In figure 15.8 the extreme points are the feasible points at which exactly three inequalities are active. Based on these first examples, one might conjecture that a feasible point of a polyhedron in $\mathbf{R}^n$ is an extreme point if exactly n inequalities are active. The polyhedron defined by (15.8), depicted in figure 15.10, shows that this is false. Consider $\hat{x} = (1/2, 1/2, 0)$. Three inequalities are active at this point:

$$-x_3 \leq 0, \qquad x_1 + x_2 + x_3 \leq 1, \qquad -x_1 - x_2 - x_3 \leq -1,$$

and although the number of active inequalities is equal to $n = 3$, the point $\hat{x}$ is not an extreme point. The algebraic definition gives the correct answer. We have $I(\hat{x}) = \{3, 4, 5\}$ and

$$A_{I(\hat{x})} = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}.$$

This is a singular 3×3 matrix (for example, the vector $(1, -1, 0)$ is in the nullspace), so $\hat{x}$ is not an extreme point.

If we apply the algebraic condition to $\hat{x} = (1, 0, 0)$, we see that there are four active constraints

$$-x_2 \leq 0, \qquad -x_3 \leq 0, \qquad x_1 + x_2 + x_3 \leq 1, \qquad -x_1 - x_2 - x_3 \leq 1,$$

and

$$A_{I(\hat{x})} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}.$$

This 4×3 matrix is left-invertible; therefore $\hat{x} = (1, 0, 0)$ is an extreme point.

Equivalence of geometric and algebraic definitions We now show that the two definitions of extreme point are equivalent. We assume that $\hat{x}$ is a feasible point at which p inequalities are active. By rearranging the rows in A and b we can assume that the first p inequalities are active, *i.e.*, $I(\hat{x}) = \{1, \ldots, p\}$. To simplify the notation we define

$$\hat{A} = A_{I(\hat{x})} = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_p^T \end{bmatrix}, \qquad \hat{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}.$$

With this notation we can write that

$$\hat{A}\hat{x} = \hat{b}, \qquad a_k^T \hat{x} < b_k, \quad k = p+1, \ldots, m.$$

We distinguish two cases.

- *$\hat{A}$ is left-invertible* ($\hat{x}$ satisfies the algebraic definition of extreme point). Let y, z be two points in $\mathcal{P}$ with $\hat{x} = (1/2)(y+z)$. The first p inequalities are active at $\hat{x}$, so we have $a_k^T \hat{x} = b_k$ or

$$\frac{1}{2}(a_k^T y + a_k^T z) = b_k \tag{15.9}$$

  for $k = 1, \ldots, p$. Combining this with $a_k^T y \leq b_k$ gives

$$a_k^T y \leq a_k^T z, \quad k = 1, \ldots, p.$$

  Combining (15.9) with $a_k^T z \leq b_k$ gives

$$a_k^T z \leq a_k^T y, \quad k = 1, \ldots, p.$$

  Hence, $a_k^T y = a_k^T z = b_k$ for $k = 1, \ldots, p$, *i.e.*,

$$\hat{A}\hat{x} = \hat{A}y = \hat{A}z = \hat{b}.$$

  If $\hat{A}$ has a zero nullspace, this implies $y = z = \hat{x}$. We conclude that if $\hat{x} = (1/2)(y+z)$ with $y, z \in \mathcal{P}$, then $y = z = \hat{x}$. This shows that $\hat{x}$ satisfies the geometric definition of extreme point.

- *$\hat{A}$ is not left-invertible* ($\hat{x}$ does not satisfy the algebraic definition of extreme point). Let v be a nonzero vector in the nullspace of $\hat{A}$. Define $y = \hat{x} + tv$ and $z = \hat{x} - tv$ where t is a positive scalar. Then $\hat{x} = (1/2)(y+z)$, so $\hat{x}$ is the midpoint of the line segment defined by y and z. For $1 \leq k \leq p$,

$$a_k^T y = a_k^T \hat{x} + t a_k^T v = a_k^T \hat{x} = b_k$$

  and similarly, $a_k^T z = b_k$. The endpoints y and z therefore satisfy the first p inequalities with equality. For the remaining inequalities we have

$$a_k^T y = a_k^T \hat{x} + t a_k^T v < b_k + t a_k^T v, \quad k = p+1, \ldots, m,$$

and

$$a_k^T z = a_k^T \hat{x} - t a_k^T v < b_k - t a_k^T v, \quad k = p+1, \ldots, m.$$

The inner product $a_k^T v$ can be nonzero, but for sufficiently small t, we have $a_k^T y \leq b_k$ and $a_k^T z \leq b_k$. Specifically, $a_k^T y \leq b_k$ and $a_k^T z \leq b_k$ if we take

$$t \leq \frac{b_k - a_k^T \hat{x}}{|a_k^T v|}$$

for all $k = p+1, \ldots, m$ with $a_k^T v \neq 0$. Therefore we can express $\hat{x}$ as the midpoint of a line segment with endpoints in $\mathcal{P}$. This shows that $\hat{x}$ does not satisfy the geometric definition of extreme point.

15.5 Simplex algorithm

The simplex algorithm solves an LP by moving from one extreme point to another with lower cost, until an optimal extreme point is found. We will make two simplifying assumptions in our presentation of the algorithm.

- *There are exactly n active constraints at each extreme point.* This means that if $\hat{x}$ is an extreme point, the matrix $A_{I(\hat{x})}$ with the coefficients of the active constraints is square and nonsingular. Such an extreme point is called a *nondegenerate* extreme point. An extreme point at which more than n inequalities are active is called *degenerate*.

  The nondegeneracy assumption is made to simplify the discussion of the simplex algorithm. However, straightforward modifications of the algorithm presented below can handle LPs with degenerate extreme points.

- *An extreme point of the polyhedron is given.* (Note that this implies that the problem is feasible, so the optimal value must be finite or $-\infty$.)

  Practical implementations of the algorithm first solve an auxiliary LP to find an extreme point. This is referred to as *phase 1* of the algorithm; the actual minimization is referred to as *phase 2*. In these notes we only discuss phase 2 and omit the details of phase 1.

The outline of the simplex algorithm is as follows.

Algorithm 15.1. SIMPLEX ALGORITHM.

given an extreme point x with active constraint set $I = I(x)$.

repeat

1. Compute $z = A_I^{-T} c$. If $z \leq 0$, terminate: x is optimal. Otherwise choose an index j with $z_j > 0$.
2. Compute $v = -A_I^{-1} e_j$. If $Av \leq 0$, terminate: the problem is unbounded below.

3. Update $x := x + t_{\max} v$ where

$$t_{\max} = \min_{i: a_i^T v > 0} \frac{b_i - a_i^T x}{a_i^T v}.$$

Set I equal to the active constraint set $I(x)$.

We will discuss the steps in the algorithm later, after going through a small example.

Example We apply the simplex algorithm to the LP

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 - x_3 \\ \\ \text{subject to} & \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 5 \end{bmatrix} \end{array} \tag{15.10}$$

(see figure 15.12). The algorithm is started at the extreme point $x = (2, 2, 0)$.

1. $x = (2, 2, 0)$. We have $b - Ax = (2, 0, 2, 0, 0, 2, 1)$, so the active constraints are 2, 4, 5:

$$I = \{2, 4, 5\}, \qquad A_I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

In step 1 we compute $z = A_I^{-T} c = (1, 1, 1)$. All three entries of z are positive, so we can choose any j. If we choose $j = 3$, we get

$$v = A_I^{-1} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

We have $Av = (0, 0, 0, 0, -1, 1, 1)$ so step 3 gives

$$t_{\max} = \min\{\frac{b_6 - a_6^T x}{a_6^T v}, \frac{b_7 - a_7^T x}{a_7^T v}\} = \min\{\frac{2}{1}, \frac{1}{1}\} = 1.$$

Taking a step $t_{\max} = 1$ in the direction v brings us to the extreme point $x + t_{\max} v = (2, 2, 1)$.

2. $x = (2, 2, 1)$. We have $b - Ax = (2, 0, 2, 0, 1, 1, 0)$, so the active constraints are 2, 4, 7:

$$I = \{2, 4, 7\}, \qquad A_I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

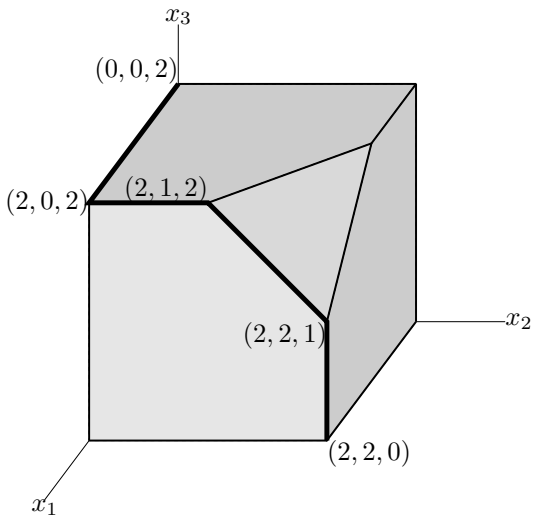


Figure 15.12 Path of the simplex algorithm applied to the LP (15.10), starting at the extreme point $(2, 2, 0)$.

Step 1 gives $z = A_I^{-T} c = (2, 2, -1)$. We can choose $j = 1$ or $j = 2$. If we choose $j = 2$, we find in step 2

$$v = A_I^{-1} \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}.$$

We have $Av = (0, 0, 1, -1, -1, 1, 0)$. Therefore step 3 gives

$$t_{\max} = \min\{\frac{b_3 - a_3^T x}{a_3^T v}, \frac{b_6 - a_6^T x}{a_6^T v}\} = \min\{\frac{2}{1}, \frac{1}{1}\} = 1.$$

The next extreme point is $x + t_{\max} v = (2, 1, 2)$.

3. $x = (2, 1, 2)$. We have $b - Ax = (2, 0, 1, 1, 2, 0, 0)$, so the active constraints are 2, 6, 7:

$$I = \{2, 6, 7\}, \qquad A_I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Step 1 gives $z = A_I^{-T} c = (0, -2, 1)$. The only possible choice for j is $j = 3$. Step 2 gives

$$v = A_I^{-1} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}.$$

We have $Av = (0, 0, 1, -1, 0, 0, -1)$. Therefore step 3 gives

$$t_{\max} = \frac{b_3 - a_3^T x}{a_3^T v} = \frac{1}{1} = 1.$$

The next extreme point is $x + t_{\max} v = (2, 0, 2)$.

4. $x = (2, 0, 2)$. We have $b - Ax = (2, 0, 0, 2, 2, 0, 1)$ so the active constraints are 2, 3, 6:
$$I = \{2, 3, 6\}, \qquad A_I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$
Step 1 gives $z = A_I^{-T} c = (1, -1, -1)$. The only possible choice for j is $j = 1$. Step 2 gives
$$v = A_I^{-1} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}.$$
We have $Av = (1, -1, 0, 0, 0, 0, -1)$, so step 3 gives
$$t_{\max} = \frac{b_1 - a_1^T x}{a_1^T v} = \frac{2}{1} = 2.$$
The next extreme point is $x + t_{\max} v = (0, 0, 2)$.

5. $x = (0, 0, 2)$. We have $b - Ax = (0, 2, 0, 2, 2, 0, 3)$. The active constraints are 1, 3, 6:
$$I = \{1, 3, 6\}, \qquad A_I = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$
Step 1 gives $z = A_I^{-T} c = (-1, -1, -1)$. All the entries of z are negative, and the algorithm terminates here. The optimal solution is $x = (0, 0, 2)$.

Discussion We now analyze one iteration of the simplex algorithm. To simplify the notation we will assume that the active set at the beginning of step 1 is $I = \{1, 2, \ldots, n\}$. If we define
$$A_I = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{bmatrix}, \qquad b_I = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$
then $A_I x = b_I$ and A_I is nonsingular. The other constraints are inactive: $a_k^T x < b_k$ for $k = n + 1, \ldots, m$,

- *Step 1.* We first verify that x is optimal if $z = A_I^{-T} c \leq 0$. Let y be any feasible point. Then
$$\begin{aligned} c^T(x - y) &= z^T A_I (x - y) \\ &= z^T (b_I - A_I y) \\ &= \sum_{k=1}^{n} z_k (b_k - a_k^T y). \end{aligned}$$

If $z \leq 0$ and y is feasible, then each term in the last expression is the product of a nonpositive number z_k and a nonnegative number $b_k - a_k^T y$. Therefore the sum is less than or equal to zero. Hence, $c^T x \leq c^T y$ for all feasible y, and x is indeed optimal.

- *Step 2.* We verify that if $Av \leq 0$ in step 2, then the problem is unbounded below. Note that for $t \geq 0$,

$$A(x + tv) = Ax + tAv \leq Ax \leq b,$$

so $x + tv$ is feasible for all $t \geq 0$. The objective value of $x + tv$ is

$$\begin{aligned} c^T(x + tv) &= c^T x + tz^T A_I v \\ &= c^T x - tz^T e_j \\ &= c^T x - tz_j. \end{aligned}$$

Since $z_j > 0$, this goes to $-\infty$ as $t \to \infty$. The algorithm has therefore identified a half-line $\{x + tv \mid t \geq 0\}$ of feasible points, along which the cost function decreases linearly with slope $-z_j$.

- *Step 3.* We will show that the point $x^+ = x + t_{\max} v$ is an extreme point with lower cost than the current extreme point x.

  The definition of $t_{\max}$ involves the ratios

$$\frac{b_i - a_i^T x}{a_i^T v}$$

  for the indices i with $a_i^T v > 0$. This step in the simplex algorithm is sometimes called the *minimum-ratio test*. Define $K = \{i = 1, \ldots, m \mid a_i^T v > 0\}$. The set K is nonempty because if $a_i^T v \leq 0$ for all i we terminate in step 2. We can also observe from step 2 that $A_I v = -e_j$, so for $i \leq n$ the inner products $a_i^T v$ are either zero or -1. Therefore K certainly does not contain any of the first n indices. The other indices ($i > n$) are indices of inactive constraints. Therefore $b_i - a_i^T x > 0$ for all $i \in K$, and $t_{\max}$ is the minimum of positive numbers.

  Next we show that $x^+ = x + t_{\max} v$ is feasible. We have

$$a_k^T x^+ = a_k^T x + t_{\max} a_k^T v.$$

  If $a_k^T v \leq 0$, then $a_k^T x^+ \leq b_k$ because $a_k^T x \leq b_k$ and $t_{\max} > 0$. If $a_k^T v > 0$, the inequality $a_k^T x^+ \leq b_k$ is satisfied because

$$t_{\max} \leq \frac{b_k - a_k^T x}{a_k^T v}$$

  by definition of $t_{\max}$. The cost function at x^+

$$\begin{aligned} c^T x + t_{\max} c^T v &= c^T x + t_{\max} z^T A_I v \\ &= c^T x - t_{\max} z_j \\ &< c^T x. \end{aligned}$$

It remains to show that x^+ is an extreme point. To do this we need to determine which constraints are active at x^+. We have

$$\begin{aligned} A_I x^+ &= A_I x + t_{\max} A_I v \\ &= b_I - t_{\max} e_j. \end{aligned}$$

Therefore $n-1$ of the first n constraints are active at x^+; constraint j is inactive. If we assume (without loss of generality) that $j = 1$, then the constraints $2, \ldots, n$ are active at x^+. The remaining constraints $k > n$ are active if $a_k^T x + t_{\max} a_k^T v = b_k$, *i.e.*, $a_k^T v > 0$ and

$$\frac{b_k - a_k^T x}{a_k^T v} = t_{\max}.$$

This is true for at least one $k > n$. Suppose (without loss of generality) that it is true for $k = n+1, \ldots, n+l$. Then the active constraints at x^+ are constraints $2, \ldots, n+l$, and we have to verify that

$$A_{I(x^+)} = \begin{bmatrix} a_2^T \\ \vdots \\ a_n^T \\ a_{n+1}^T \\ \vdots \\ a_{n+l}^T \end{bmatrix}$$

is left-invertible. The first $n-1$ rows of this matrix are linearly independent because they are rows of the square nonsingular matrix A_I. The vector a_{n+1} is also independent of $a_2, \ldots, a_n$ because $a_2^T v = \cdots = a_n^T v = 0$ and $a_{n+1}^T v > 0$. Therefore $A_{I(\hat{x})}$ has n independent rows and its nullspace is zero. We can conclude that x^+ is an extreme point.

Since we assumed that all extreme points are nondegenerate, we know that x^+ has exactly n active constraints, *i.e.*, $l = 1$.

Complexity Establishing convergence of the simplex algorithm is trivial. A polyhedron has a finite number of extreme points (bounded above by $m!/((m-n)!n!)$, and since the cost function decreases strictly at each iteration, the algorithm terminates after a finite number of steps.

The worst-case complexity is very high, because the number of extreme points of a general LP can be astronomically large and in the worst case the algorithm visits all the extreme points before finding the optimal one. In practice, however, the algorithm turns out to be very efficient and the number of steps is generally observed to be bounded by a linear function of the problem dimensions.

Each step of the algorithm involves the solution of two equations

$$A_I^T z = c, \qquad A_I v = -e_j.$$

As we have seen in chapter 7 this requires only one LU factorization of A_I. Practical problems are often very large and sparse, and a sparse LU factorization is used to solve the two equations.

A further improvement follows from the fact that at each iteration the matrix A_I is modified by replacing one row with a new row (the jth row selected in step 1 is replaced with the row of A that achieves the minimum in step 3). Exploiting this property makes it possible to obtain the LU factors of A_I by updating the LU factorization of the previous step. For dense A the cost of the update is order n^2 operations, as opposed to $(2/3)n^3$ for a new factorization.

Exercises

15.1 Let P be the set of vectors $x = (x_1, x_2, x_3, x_4)$ in $\mathbf{R}^4$ that satisfy the conditions

$$x_2 + x_3 + x_4 = 1, \qquad 2x_1 + x_3 = 1, \qquad x_1 \geq 0, \qquad x_2 \geq 0, \qquad x_3 \geq 0, \qquad x_4 \geq 0.$$

(a) Write P as a the solution set of linear inequalities $Ax \leq b$.

(b) Which of the following three vectors are extreme points of P?

$$x = (1/2,\, 0,\, 0,\, 1), \qquad x = (0,\, 0,\, 1,\, 0), \qquad x = (1/4,\, 1/2,\, 1/2,\, 0).$$

Part IV

Accuracy of numerical algorithms

Chapter 16

Conditioning and stability

The final two chapters are a short introduction to topics related to the accuracy of numerical algorithms. In this chapter we discuss the concepts of problem conditioning and algorithm stability. These two concepts are related, but it is important to distinguish between them: *conditioning* is a property of a mathematical problem and has nothing to do with how the problem is solved; *stability* is a property of an algorithm for solving a problem.

16.1 Problem conditioning

A mathematical problem is *well conditioned* if small changes in the problem parameters (the problem data) lead to small changes in the solution. A problem is *badly conditioned* or *ill-conditioned* if small changes in the parameters can cause large changes in the solution. In other words, the solution of a badly conditioned problem is very sensitive to changes in the parameters. Note that this is an informal definition. A precise definition requires defining what we mean by large or small errors (*e.g.*, relative or absolute error, choice of norm), and a statement of which of the parameters are subject to error.

In engineering problems, the data are almost always subject to uncertainty, due to measurement errors, imperfect knowledge of the system, modeling approximations, rounding errors in previous calculations, etc., so it is important in practice to have an idea of the conditioning of a problem.

Example Consider the two equations in two variables

$$\begin{aligned} x_1 + x_2 &= b_1 \\ (1 + 10^{-5})x_1 + (1 - 10^{-5})x_2 &= b_2. \end{aligned}$$

It is easily verified that the coefficient matrix

$$A = \begin{bmatrix} 1 & 1 \\ 1 + 10^{-5} & 1 - 10^{-5} \end{bmatrix}$$

is nonsingular, with inverse

$$A^{-1} = \frac{1}{2}\begin{bmatrix} 1-10^5 & 10^5 \\ 1+10^5 & -10^5 \end{bmatrix},$$

so the solution of $Ax = b$ is

$$x = A^{-1}b = \frac{1}{2}\begin{bmatrix} 1-10^5 & 10^5 \\ 1+10^5 & -10^5 \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} b_1 - 10^5(b_1-b_2) \\ b_1 + 10^5(b_1-b_2) \end{bmatrix}.$$

Plugging in a few values for b_1 and b_2 around $(1, 1)$ gives the following results.

b_1	b_2	x_1	x_2
1.00	1.00	0.500	0.500
1.01	1.01	0.505	0.505
0.99	1.01	1000.495	−999.505
1.01	0.99	−999.505	1000.495
0.99	0.99	0.495	0.495

We immediately notice that small changes in b sometimes lead to very large changes in the solution x. In most applications that would pose a very serious problem. Suppose for example that the values $b_1 = 1$ and $b_2 = 1$ are obtained from measurements with a precision of 1%, so the actual values of b_1 and b_2 can be anywhere in the interval $[0.99, 1.01]$. The five values of x_1 and x_2 in the table are all in agreement with the measurements, so it would be foolish to accept the value $x_1 = x_2 = 0.5$, obtained from the measured values $b_1 = b_2 = 1$, as the correct solution.

A second observation is that the error in x is not always large. In the second and fifth rows of the table, the error in x is 1%, of the same order as the error in b.

In this example we only changed the values of the right-hand side b, and assumed that the matrix A is exactly known. We can also consider the effect of small changes in the coefficients of A, with similar conclusions. In fact, A is very close to the singular matrix

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

so small changes in A might result in an unsolvable set of equations.

16.2 Condition number

The most common method for examining the conditioning of a problem is precisely what we did in the small example above. We generate a number of possible values of the problem data, solve the problem for each of those values, and compare the results. If the results vary widely, we conclude the problem is ill-conditioned. If the results are all close, we can say with some confidence that the problem is probably well-conditioned. This method is simple and applies to any type of problem, not just linear equations. It can also be quite dangerous. For example, if in the small example above, we had solved the equations for three right-hand sides,

$(b_1, b_2) = (1, 1)$, $(b_1, b_2) = (1.01, 1.01)$, $(b_1, b_2) = (0.99, 0.99)$, then we would have incorrectly concluded that the problem is well-conditioned.

A more rigorous method is to mathematically derive bounds on the error in the solution, given a certain error in the problem data. For most numerical problems this is quite difficult, and it is the subject of a field of mathematics called Numerical Analysis. In this course we will give one example of such an error analysis: we will derive simple and easily computed bounds on the error of the solution of a set of linear equations, given an error in the right-hand side.

Suppose we are given a set of linear equations $Ax = b$, with A nonsingular and of order n. The solution x exists and is unique, and can be expressed as $x = A^{-1}b$. Now suppose we replace b with $b + \Delta b$. The new solution is

$$x + \Delta x = A^{-1}(b + \Delta b) = A^{-1}b + A^{-1}\Delta b = x + A^{-1}\Delta b,$$

so $\Delta x = A^{-1}\Delta b$. We are interested in deriving bounds on Δx, given bounds on Δb.

We discuss two types of bounds: one relating the *absolute errors* $\|\Delta x\|$ and $\|\Delta b\|$ (measured in Euclidean norm), the second relating the *relative errors* $\|\Delta x\|/\|x\|$ and $\|\Delta b\|/\|b\|$.

Absolute error bounds Using the fifth property of matrix norms on page 40, we find the following bound on $\|\Delta x\| = \|A^{-1}\Delta b\|$:

$$\|\Delta x\| \leq \|A^{-1}\| \, \|\Delta b\|. \tag{16.1}$$

This means that if $\|A^{-1}\|$ is small, then small changes in the right-hand side b always result in small changes in x. On the other hand if $\|A^{-1}\|$ is large, then small changes in b may result in large changes in x.

Relative error bounds We find a bound on the relative error by combining (16.1) and $\|b\| \leq \|A\| \, \|x\|$ (which follows from $b = Ax$ and property 5):

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A\| \, \|A^{-1}\| \frac{\|\Delta b\|}{\|b\|}. \tag{16.2}$$

The product $\|A\| \, \|A^{-1}\|$ is called the *condition number* of A, and is denoted $\kappa(A)$:

$$\kappa(A) = \|A\| \, \|A^{-1}\|.$$

Using this notation we can write the inequality (16.2) as

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\Delta b\|}{\|b\|}.$$

It follows from the sixth property of matrix norms on page 40 that

$$\kappa(A) = \|A\|\|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1$$

for all nonsingular A. A small condition number (*i.e.*, $\kappa(A)$ close to one), means that the relative error in x is not much larger than the relative error in b. We say that the matrix A is *well-conditioned* if its condition number is small. A large condition number means that the relative error in x may be much larger than the relative error in b. If $\kappa(A)$ is large, we say that A is *ill-conditioned*.

Computing the condition number The MATLAB command `cond(A)` computes the condition number of a matrix A (using a method that we will not cover). The commands `rcond(A)` and `condest(A)` provide very fast *estimates* of the condition number or its inverse. Fast condition number estimators are useful for large matrices, when calculating the exact condition number is too expensive.

16.3 Algorithm stability

Numerical algorithms almost never compute the exact solution of a mathematical problem, but only a very good approximation. The most important source of error is rounding error, introduced by the finite precision used by computers. The result of an algorithm may also be inaccurate because the method is based on discretization (*e.g.*, in numerical integration, or when solving differential equations), or truncation (*e.g.*, evaluating a function based on the Taylor series).

An algorithm is *numerically stable* if the inevitable small errors introduced during the calculations lead to small errors in the result. It is *unstable* if small errors during the calculation can lead to very large errors in the result. Again, this is an informal definition, but sufficient for our purposes.

We have already encountered two examples of instability. In section 7.7 we noted that solving linear equations via the LU factorization without row permutations is unstable. In section 9.5 we noted that the QR factorization method for solving least-squares problems is more stable than the Cholesky factorization method. We will give other examples of instability in section 16.4.

Note the difference between conditioning and stability. Conditioning is a property of a *problem*, while stability is a property of an *algorithm*. If a problem is badly conditioned and the parameters are subject to error, then the solution will be inaccurate, regardless of how it is computed. If an algorithm is unstable, then the result is inaccurate because the algorithm introduces 'unnecessarily' large errors.

In practice, of course, we should always try to use stable algorithms, while the conditioning of a problem is not always under our control.

16.4 Cancellation

Instability in an algorithm is often (but not always) caused by an effect called *cancellation*. Cancellation occurs when two numbers are subtracted that are almost equal, and one of the numbers or both are subject to error (for example, due to rounding error in previous calculations).

Suppose

$$\hat{x} = x + \Delta x, \qquad \hat{y} = y + \Delta y$$

are approximations of two numbers x, y, with absolute errors $|\Delta x|$ and $|\Delta y|$, re-

spectively. The relative error in the difference $\hat{x} - \hat{y}$ is

$$\frac{|(\hat{x} - \hat{y}) - (x - y)|}{|x - y|} = \frac{|\Delta x - \Delta y|}{|x - y|} \leq \frac{|\Delta x| + |\Delta y|}{|x - y|}.$$

(The upper bound is achieved when Δx and Δy have opposite signs.) We see that if $x - y$ is small, then the relative error in $\hat{x} - \hat{y}$ can be very large, and much larger than the relative errors in $\hat{x}$ and $\hat{y}$. The result is that the relative errors $|\Delta x|/|x|$, $|\Delta y|/|y|$ are magnified enormously.

For example, suppose $x = 1$, $y = 1+10^{-5}$, and x and y have been calculated with an accuracy of about 10 significant digits, *i.e.*, $|\Delta x|/|x| \approx 10^{-10}$ and $|\Delta y|/|y| \approx 10^{-10}$. The error in the result is

$$\frac{|(\hat{x} - \hat{y}) - (x - y)|}{|x - y|} \leq \frac{|\Delta x| + |\Delta y|}{|x - y|} \approx \frac{2 \cdot 10^{-10}}{|x - y|} = 2 \cdot 10^{-5}.$$

The result has only about 5 correct digits.

Example The most straightforward method for computing the two roots of the quadratic equation

$$ax^2 + bx + c = 0$$

(with $a \neq 0$) is to evaluate the expressions

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

This method is unstable if $b^2 \gg |4ac|$. If $b > 0$, there is a danger of cancellation in the expression for x_1; if $b < 0$, cancellation may occur in the expression for x_2.

For example, suppose $a = c = 1$, $b = 10^5 + 10^{-5}$. The exact roots are given by

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = -10^{-5}, \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = -10^5,$$

We evaluate these expressions in MATLAB, rounding the square roots to 6 correct digits[1]:

```
>> a = 1;  b = 1e5 + 1e-5;  c = 1;
>> x1 = (-b + chop(sqrt(b^2 - 4*a*c),6))/(2*a)
ans =
   -5.0000e-6
>> x2 = (-b - chop(sqrt(b^2 - 4*a*c),6))/(2*a)
ans =
   -1.0000e+05
```

The relative error in x_1 is 50%, and is due to cancellation.

[1] We use the MATLAB function `chop`; Octave users can download `chop.m` from the course website.

We can formulate an algorithm that is more stable if $b^2 \gg |4ac|$ as follows. First suppose $b > 0$, so we have cancellation in the expression for x_1. In this case we can calculate x_2 accurately. The expression for x_1 can be reformulated as

$$\begin{aligned} x_1 &= \frac{\left(-b + \sqrt{b^2 - 4ac}\right)\left(-b - \sqrt{b^2 - 4ac}\right)}{(2a)\left(-b - \sqrt{b^2 - 4ac}\right)} \\ &= \frac{b^2 - b^2 + 4ac}{(2a)(-b - \sqrt{b^2 - 4ac})} \\ &= \frac{c}{ax_2}. \end{aligned}$$

Similarly, if $b > 0$, we can use the expression $x_2 = c/(ax_1)$ to compute x_2, given x_1. The modified algorithm that avoids cancellation is therefore:

- if $b \leq 0$, calculate

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{c}{ax_1}$$

- if $b > 0$, calculate

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \qquad x_1 = \frac{c}{ax_2}.$$

For the example, we get

```
>> a = 1;  b = 1e5 + 1e-5;  c = 1;
>> x2 = (-b - chop(sqrt(b^2 - 4*a*c),6))/(2*a)
ans =
   -1.0000e+05
>> x1 = c/(a*x2)
ans =
   -1.0000e-05
```

Exercises

Conditioning of a set of linear equations

16.1

$$A = \begin{bmatrix} 0 & 0 & -10^4 & 0 \\ 0 & 0 & 0 & -10 \\ 0 & 10^{-3} & 0 & 0 \\ 10^{-2} & 0 & 0 & 0 \end{bmatrix}.$$

(a) What is the norm of A?

(b) What is the inverse of A?

(c) What is the norm of the inverse of A?

(d) What is the condition number of A?

Explain your answers, without referring to any MATLAB results. (Of course, you are free to check the answers in MATLAB.)

16.2 Give the matrix norm $\|A\|$ of each of the following matrices A, without using MATLAB. If A is nonsingular, also give $\|A^{-1}\|$ and $\kappa(A)$.

(a) $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

(b) $A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$

(c) $A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & -3/2 \end{bmatrix}$

(d) $A = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & -3/2 \end{bmatrix}$

(e) $A = \begin{bmatrix} 0 & -1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & -3 \end{bmatrix}$

(f) $A = \begin{bmatrix} 2 & -1 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -3 \end{bmatrix}$

(g) $A = \begin{bmatrix} 2 & -1 & -3 \\ -2 & 1 & 3 \\ 2 & -1 & -3 \end{bmatrix}$

16.3 Suppose Q is square and orthogonal. For each of the following matrices A, give $\|A\|$ and, if A is invertible, also $\|A^{-1}\|$. Explain your answers.

(a) $A = \begin{bmatrix} Q & -Q \\ Q & Q \end{bmatrix}$.

(b) $A = \begin{bmatrix} Q & Q \\ Q & Q \end{bmatrix}$.

16.4 The table shows $\|Ax^{(i)}\|$ and $\|x^{(i)}\|$ for four vectors $x^{(1)}$, $x^{(2)}$, $x^{(3)}$, $x^{(4)}$, where A is a nonsingular $n \times n$-matrix.

	$\|x^{(i)}\|$	$\|Ax^{(i)}\|$
$i = 1$	1	100
$i = 2$	100	1
$i = 3$	10^3	10^4
$i = 4$	10^{-3}	10^2

What are the best (*i.e.*, greatest) lower bounds on $\|A\|$, $\|A^{-1}\|$ and $\kappa(A)$ that you can derive based on this information?

16.5 Suppose A is a nonsingular 4×4 matrix with columns a_1, a_2, a_3, a_4:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix}.$$

We are given the norms of the column vectors:

$$\|a_1\| = 1, \qquad \|a_2\| = 10^4, \qquad \|a_3\| = 10^{-3}, \qquad \|a_4\| = 10^{-3}.$$

Based on this information, what can you say about

(a) the norm of A

(b) the norm of A^{-1}

(c) the condition number of A?

Be as precise as possible. If you can derive the exact value (for example, $\|A\| = 2.3$), give the exact value. Otherwise give an upper bound (for example, $\|A\| \leq 2.3$) or a lower bound (for example, $\|A\| \geq 2.3$). Upper bounds are more precise if they are lower ($\|A\| \leq 2.3$ is a more interesting bound than $\|A\| \leq 5$). Lower bounds are more precise if they are higher ($\|A\| \geq 2.3$ is a more interesting bound then $\|A\| \geq 1.5$).

16.6 Consider a set of linear equations $Ax = b$ with

$$A = \begin{bmatrix} 1 + 10^{-8} & 1 \\ 1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

It is easily verified that A is nonsingular with inverse

$$A^{-1} = 10^8 \begin{bmatrix} 1 & -1 \\ -1 & 1 + 10^{-8} \end{bmatrix}.$$

(a) Prove, without using MATLAB, that $\kappa(A) \geq 10^8$.

(b) Find the solution of $Ax = b$. Construct a perturbation Δb of the right-hand side for which we have

$$\frac{\|\Delta x\|}{\|x\|} \geq 10^8 \frac{\|\Delta b\|}{\|b\|},$$

where $x + \Delta x$ is the solution of the equations $A(x + \Delta x) = b + \Delta b$.

16.7 Sort the following matrices in order of decreasing condition number (without using MATLAB):

$$A_1 = \begin{bmatrix} 10^5 & 1 \\ 1 & -10^5 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} 10^5 & 1 \\ 1 & -10^{-5} \end{bmatrix},$$

$$A_3 = \begin{bmatrix} 10^{-5} & 1 \\ 1 & -10^{-5} \end{bmatrix}, \qquad A_4 = \begin{bmatrix} 10^5 & 1 \\ 1 & 10^{-5} \end{bmatrix}.$$

If any of the matrices is singular, take ∞ as its condition number. Explain your answer.

16.8 *Condition numbers and diagonal scaling.* The matrix A that represents a linear function $y = Ax$ depends on the units we choose for x and y. Suppose for example that the components of x are currents and the components of y are voltages, and that we have $y = Ax$ with x in amperes and y in volts. Now suppose $\tilde{x}_1$ is x_1 expressed in milliamperes, *i.e.*, $\tilde{x}_1 = 1000x_1$. Then we can write

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11}/1000 & a_{12} & \cdots & a_{1n} \\ a_{21}/1000 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}/1000 & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = AD \begin{bmatrix} \widetilde{x}_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where D is a diagonal matrix with diagonal elements $d_{11} = 1/1000$, $d_{22} = \cdots = d_{nn} = 1$. Likewise, if $\tilde{y}_1$ is y_1 expressed in millivolts, we have

$$\begin{bmatrix} \tilde{y}_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1000\,a_{11} & 1000\,a_{12} & \cdots & 1000\,a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = DAx$$

where D is a diagonal matrix with diagonal elements $d_{11} = 1000$, $d_{22} = \cdots = d_{nn} = 1$.

In general, changing the units for x corresponds to replacing A with AD where D is positive diagonal matrix; changing the units for y corresponds to replacing A with DA where D is positive and diagonal.

In this problem we examine the effect of scaling columns or rows of a matrix on its condition number.

(a) Prove the following properties of the matrix norm. We assume A is $n \times n$ with

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix},$$

i.e., a_i stands for the ith column of A.

(a) $\|A\| \geq \max_{i=1,\ldots,n} \|a_i\|$ (*i.e.*, the norm of A is greater than or equal to the norm of each column a_i)

(b) $\|A^{-1}\| \geq 1/\min_{i=1,\ldots,n} \|a_i\|$ (*i.e.*, $1/\|A^{-1}\|$ is less than or equal to the norm of each column a_i)

(c) The condition number satisfies the lower bound

$$\kappa(A) \geq \frac{\max_{i=1,\ldots,n} \|a_i\|}{\min_{i=1,\ldots,n} \|a_i\|}.$$

In other words, if the columns of A are very different in norm, (*i.e.*, $\max_i \|a_i\| \gg \min_i \|a_i\|$), then A will certainly have a large condition number. In practice, it is therefore recommended to scale the columns of a matrix so that they are approximately equal in norm. Similar comments apply to row scaling.

(b) As an example, consider the matrix

$$A = \begin{bmatrix} 1 & 3 \cdot 10^{-3} & 11 \\ -2 \cdot 10^{-2} & 10^5 & -4 \cdot 10^2 \\ 1 & 10^4 & 10^8 \end{bmatrix}.$$

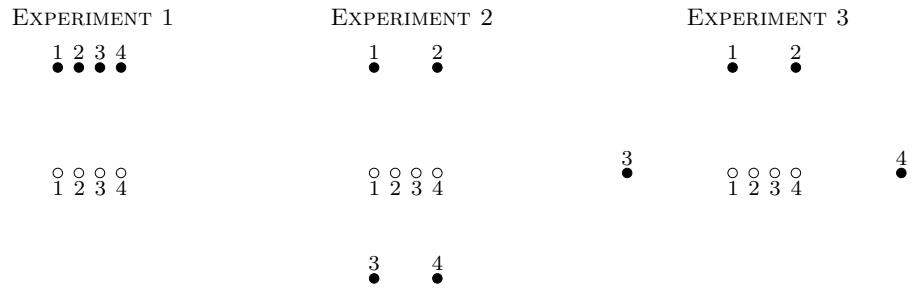
(a) Determine the condition number of A (using MATLAB's `cond` function).

(b) Find a diagonal matrix D such that all columns of $\widetilde{A} = AD$ have the same norm. Determine the condition number of $\widetilde{A}$.

16.9 The figure below shows three possible experiments designed to estimate the magnitudes of signals emitted by four sources. The location of the sources is indicated by the empty circles. The solid circles show the location of the sensors. The output y_i of sensor i is given by

$$y_i = x_1/r_{i1}^2 + x_2/r_{i2}^2 + x_3/r_{i3}^2 + x_4/r_{i4}^2$$

where x_j is the (unknown) magnitude of the signal emitted by source j and r_{ij} is the (given) distance from source j to sensor i.



For each of the three configurations, we can determine the distances r_{ij} and write these equations as

$$Ax = y$$

where

$$A = \begin{bmatrix} 1/r_{11}^2 & 1/r_{12}^2 & 1/r_{13}^2 & 1/r_{14}^2 \\ 1/r_{21}^2 & 1/r_{22}^2 & 1/r_{23}^2 & 1/r_{24}^2 \\ 1/r_{31}^2 & 1/r_{32}^2 & 1/r_{33}^2 & 1/r_{34}^2 \\ 1/r_{41}^2 & 1/r_{42}^2 & 1/r_{43}^2 & 1/r_{44}^2 \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}. \tag{16.3}$$

From the measured sensor outputs y we can then determine x by solving the equations $Ax = y$.

There is a measurement error Δy in the sensor readings, which can be as large as 0.01%, *i.e.*, $\|\Delta y\|/\|y\| \leq 10^{-4}$. From the analysis in section 16.2, we have the following bound on the relative error in x:

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\Delta y\|}{\|y\|} \leq \kappa(A) \cdot 10^{-4}$$

where $\kappa(A)$ is the condition number of A.

(a) Download the MATLAB file `ch16ex9.m` from the class webpage and execute it in MATLAB as `[A1,A2,A3] = ch16ex9`. The three matrices A_1, A_2, A_3 are the values of the matrix A in (16.3) for each of the three experiments.

Compute the condition numbers of A_1, A_2, A_3 using the MATLAB command `cond(A)`.

(b) Based on the results of part (a), which configuration would you prefer?

(c) Can you give an intuitive argument for your conclusion in part (b), based on the configuration of the sensors and sources?

16.10 We have encountered the matrix

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-2} & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-2} & t_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & t_{n-1} & t_{n-1}^2 & \cdots & t_{n-1}^{n-2} & t_{n-1}^{n-1} \\ 1 & t_n & t_n^2 & \cdots & t_n^{n-2} & t_n^{n-1} \end{bmatrix}$$

in the context of polynomial interpolation (exercise 3.1). A matrix of this form is often badly conditioned. As an example, suppose

$$t_1 = 1, \qquad t_2 = 2, \qquad t_3 = 3, \qquad \ldots, \qquad t_{n-1} = n-1, \qquad t_n = n.$$

Show that $\kappa(A) \geq n^{n-3/2}$.

16.11 Consider the matrix

$$A = \begin{bmatrix} 1+\epsilon & 1 & 2 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

(a) Show that A is singular for $\epsilon = 0$. Verify that for $\epsilon \neq 0$, the inverse is given by

$$A^{-1} = \frac{1}{\epsilon} \begin{bmatrix} 1 & 1 & -2 \\ 1 & 1-\epsilon & -2 \\ -1 & -1 & 2+\epsilon \end{bmatrix}.$$

(b) Prove that $\kappa(A) \geq 1/|\epsilon|$ if $\epsilon \neq 0$.

(c) Show with an example that the set of linear equations

$$Ax = b, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

is badly conditioned when ϵ is small (and nonzero). More specifically, give a $\Delta b \neq 0$ such that

$$\frac{\|\Delta x\|}{\|x\|} \geq \frac{1}{|\epsilon|} \frac{\|\Delta b\|}{\|b\|}$$

where x is the solution of $Ax = b$ and $x + \Delta x$ is the solution of $A(x + \Delta x) = b + \Delta b$.

16.12 Suppose A is a nonsingular $n \times n$ matrix. We denote by α_k the (Euclidean) norm of the kth column of A and by β_i the (Euclidean) norm of the ith row of A:

$$\alpha_k = \sqrt{\sum_{i=1}^n a_{ik}^2}, \qquad \beta_i = \sqrt{\sum_{k=1}^n a_{ik}^2}.$$

Show that

$$\kappa(A) \geq \frac{\max\{\alpha_{\max}, \beta_{\max}\}}{\min\{\alpha_{\min}, \beta_{\min}\}},$$

where

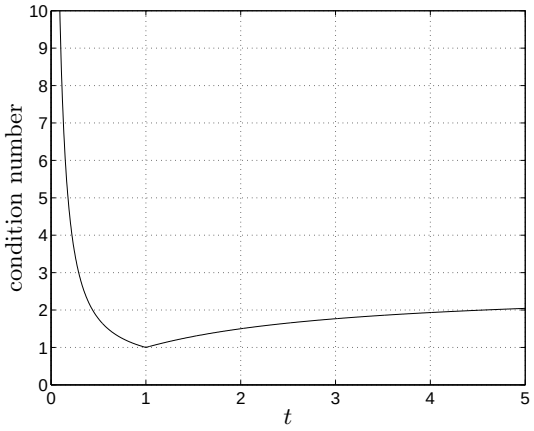
$$\alpha_{\max} = \max_{k=1,\ldots,n} \alpha_k, \qquad \alpha_{\min} = \min_{k=1,\ldots,n} \alpha_k, \qquad \beta_{\max} = \max_{i=1,\ldots,n} \beta_i, \qquad \beta_{\min} = \min_{i=1,\ldots,n} \beta_i.$$

16.13 Let L be a nonsingular $n \times n$ lower triangular matrix with elements L_{ij}. Show that

$$\kappa(L) \geq \frac{\max_{i=1,\ldots,n} |L_{ii}|}{\min_{j=1,\ldots,n} |L_{jj}|}.$$

16.14 The graph shows the condition number of one of the following matrices as a function of t for $t \geq 0$.

$$A_1 = \begin{bmatrix} t & 1 \\ 1 & -t \end{bmatrix}, \quad A_2 = \begin{bmatrix} t & t \\ -t & 1 \end{bmatrix}, \quad A_3 = \begin{bmatrix} t & 0 \\ 0 & 1+t \end{bmatrix}, \quad A_4 = \begin{bmatrix} t & -t \\ -t & 1 \end{bmatrix}.$$



Which of the four matrices was used in the figure? Carefully explain your answer.

16.15 We define A as the $n \times n$ lower triangular matrix with diagonal elements 1, and elements -1 below the diagonal:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 0 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix}.$$

(a) What is A^{-1}?

(b) Show that $\kappa(A) \geq 2^{n-2}$. This means that the matrix is ill-conditioned for large n.

(c) Show with an example that small errors in the right-hand side of $Ax = b$ can produce very large errors in x. Take $b = (0, 0, \ldots, 0, 1)$ (the n-vector with all its elements zero, except the last element, which is one), and find a nonzero Δb for which

$$\frac{\|\Delta x\|}{\|x\|} \geq 2^{n-2} \frac{\|\Delta b\|}{\|b\|},$$

where x is the solution of $Ax = b$ and $x + \Delta x$ is the solution of $A(x + \Delta x) = b + \Delta b$.

Stability and cancellation[2]

16.16 If we evaluate

$$\frac{1 - \cos x}{\sin x} \tag{16.4}$$

at $x = 10^{-2}$, rounding the cosine and sine to 4 significant digits using the `chop` command, we obtain

[2]The MATLAB command `chop(x,n)` rounds the number x to n decimal digits. We use it to artificially introduce rounding errors. For example, `chop(pi,4)` returns the number 3.14200000000000. Octave users can download the file `chop.m` from the class website.

```
>> (1-chop(cos(1e-2),4))/chop(sin(1e-2),4)
ans =
    0
```

The first four digits of the correct answer are $5.000\ 10^{-3}$. The large error is due to cancellation in the numerator $1 - \cos(x)$: $\cos 10^{-2} = 0.99995000\ldots$, so rounding to four digits yields one, and subtracting from one yields zero.

Rewrite the expression (16.4) in a form that is mathematically equivalent, but avoids cancellation. Evaluate the stable formula (still rounding cosines and sines to 4 significant digits) and compare with the result above.

16.17 You may be familiar with the following formulas for estimating the mean and the variance of a random real variable x, given n samples x_1, x_2, ..., x_n. The sample mean $\overline{x}$ (*i.e.*, the estimate of the expected value $\mathbf{E}\,x$) is given by

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{16.5}$$

The sample variance s^2 (*i.e.*, the estimate of $\mathbf{E}(x - \mathbf{E}\,x)^2$) is given by

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2, \tag{16.6}$$

which can be simplified as

$$\begin{aligned} s^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i^2 - 2x_i\overline{x} + \overline{x}^2) \\ &= \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i\overline{x} + n\overline{x}^2\right) \\ &= \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right). \end{aligned} \tag{16.7}$$

The following MATLAB code evaluates the expressions (16.5) and (16.7), using the `chop` function to simulate a machine with a precision of 6 decimal digits.

```
>> n = length(x);
>> sum1 = 0;
>> sum2 = 0;
>> for i=1:n
     sum1 = chop(sum1 + x(i)^2, 6);
     sum2 = chop(sum2 + x(i), 6);
   end;
>> samplemean = chop(sum2/n, 6)
>> samplevariance = chop((sum1 - sum2^2/n)/(n-1), 6)
```

If we run this code with input vector

```
x =  [ 1002 1000 1003 1001 1002 1002 1001 1004 1002 1001 ]
```

it returns

```
samplemean =

  1.0018e+003

samplevariance =

  -3.6000
```

This is clearly wrong, because the variance must be a nonnegative number. (The correct answer is $\overline{x} = 1001.8$, $s^2 = 1.2889$.)

(a) Explain why the result is wrong.

(b) How would you compute the sample variance more accurately, without increasing the number of correct digits in the calculation (*i.e.*, if you still round all the intermediate results to 6 decimal digits)?

16.18 It can be shown that

$$\sum_{k=1}^{\infty} k^{-2} = \pi^2/6 = 1.644934\ldots.$$

Suppose we evaluate the first 3000 terms of the sum in MATLAB, rounding the result of each addition to four digits:

```
>> sum = 0;
>> for i = 1:3000
     sum = chop(sum + 1/i^2, 4);
   end
>> sum

sum =

    1.6240
```

The result has only 2 correct significant digits.

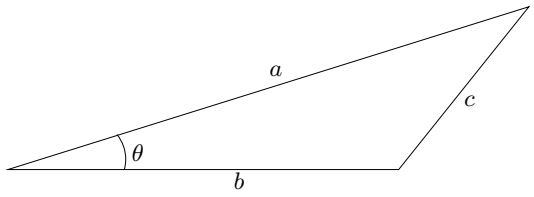
We can also evaluate the sum in the reverse order.

```
>> sum = 0;
>> for i = 3000:-1:1
     sum = chop(sum + 1/i^2, 4);
   end
>> sum

sum =

     1.6450
```

The result has four correct significant digits. Explain the difference. (Note that the calculation does not involve any subtractions, so this is an example of a large error that is *not* caused by cancellation.)

16.19 The length of one side of a triangle (c in the figure) can be calculated from the lengths of the two other sides (a and b) and the opposing angle (θ) by the formula

$$c = \sqrt{a^2 + b^2 - 2ab\cos\theta}. \tag{16.8}$$



Two equivalent expressions are

$$c = \sqrt{(a-b)^2 + 4ab\,(\sin(\theta/2))^2} \tag{16.9}$$

and

$$c = \sqrt{(a+b)^2 - 4ab\,(\cos(\theta/2))^2}. \tag{16.10}$$

(The equivalence of the three formulas follows from the identities $\cos\theta = 1 - 2\,(\sin(\theta/2))^2$ and $\cos\theta = -1 + 2\,(\cos(\theta/2))^2$.)

Which of the three formulas gives the most stable method for computing c if $a \approx b$ and θ is small? For simplicity you can assume that the calculations are exact, except for a small error in the evaluation of the cosine and sine functions. Explain your answer.

Chapter 17

Floating-point numbers

17.1 IEEE floating-point numbers

Binary floating-point numbers Modern computers use a binary floating-point format to represent real numbers. We use the notation

$$x = \pm(.d_1 d_2 \ldots d_n)_2 \cdot 2^e \tag{17.1}$$

to represent a real number x in binary floating-point notation. The first part, $.d_1 d_2 \ldots d_n$, is called the *mantissa*, and d_i is called the ith bit of the mantissa. The first bit d_1 is always equal to 1; the other $n-1$ bits can be 1 or 0. The number of bits in the mantissa, n, is called the *mantissa length*. The exponent e is an integer that can take any value between some minimum $e_{\min}$ and a maximum $e_{\max}$.

The notation (17.1) represents the number

$$x = \pm(d_1 2^{-1} + d_2 2^{-2} + \cdots + d_n 2^{-n}) \cdot 2^e.$$

For example, the number 12.625 can be written as

$$12.625 = (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} + 0 \cdot 2^{-8}) \cdot 2^4$$

and therefore its binary representation (with mantissa length 8) is

$$12.625 = +(.11001010)_2 \cdot 2^4.$$

Example A binary floating-point number system is specified by three numbers: the mantissa length n, the maximum exponent $e_{\max}$, and the minimum exponent $e_{\min}$. As a simple example, suppose $n = 3$, $e_{\min} = -1$ and $e_{\max} = 2$. Figure 17.1 shows all possible positive numbers in this number system.

We can make several interesting observations. First, it is clear that there are only finitely many numbers (16 positive and 16 negative numbers). The smallest number is

$$+(.100)_2 \cdot 2^{-1} = 0.25,$$

and the largest number is

$$+(.111)_2 \cdot 2^2 = (2^{-1} + 2^{-2} + 2^{-3}) \cdot 2^2 = 3.5.$$

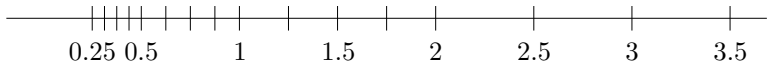


Figure 17.1 Positive numbers in a binary floating-point system with $n = 3$, $e_{\min} = -1$, $e_{\max} = 2$.

Second, the spacing between the numbers is not constant. There are four numbers with exponent $e = -1$:

$$+(.100)_2 \cdot 2^{-1} = 0.25, \quad +(.101)_2 \cdot 2^{-1} = 0.3125,$$
$$+(.110)_2 \cdot 2^{-1} = 0.375, \quad +(.111)_2 \cdot 2^{-1} = 0.4375,$$

and the difference between these numbers is $2^{-4} = 0.0625$. The next four numbers are

$$+(.100)_2 \cdot 2^{0} = 0.5, \quad +(.101)_2 \cdot 2^{0} = 0.625,$$
$$+(.110)_2 \cdot 2^{0} = 0.75, \quad +(.111)_2 \cdot 2^{0} = 0.875,$$

with a spacing of $2^{-3} = 0.125$. These are followed by the four numbers 1, 1.25, 1.5 and 1.75 (with a spacing of $2^{-2} = 0.25$) and the four numbers 2, 2.5, 3 and 3.5 (with a spacing of $2^{-1} = 0.5$).

Finally, we note that the number 0 cannot be represented, because the first bit d_1 is normalized to be 1. In practical implementations, the number 0 is represented by giving the exponent e a special value (see below).

The IEEE standard Almost all computers implement the so-called IEEE floating-point standard (published in 1985). The standard defines two types of floating-point numbers: single and double. Since numerical computations almost always use the double format, we will restrict the discussion to IEEE double precision numbers.

The IEEE double format is a binary floating-point system with

$$n = 53, \qquad e_{\min} = -1021, \qquad e_{\max} = 1024.$$

To represent an IEEE double precision number we need 64 bits: one sign bit, 52 bits for the mantissa (recall that the first bit is always one, so we do not have to store it), and 11 bits to represent the exponent. Note that the bit string for the exponent can actually take $2^{11} = 2048$ different values, while only 2046 values are needed to represent all integers between -1021 and 1024. The two remaining values are given a special meaning. One value is used to represent *subnormal* numbers, *i.e.*, small numbers with first bit $d_1 = 0$, including the number 0. The other value represents $\pm\infty$ or NaN, depending on the value of the mantissa. The value $\pm\infty$ indicates overflow; NaN (not a number) indicates arithmetic error.

17.2 Machine precision

The mantissa length n is by far the most important of the three numbers n, $e_{\min}$, $e_{\max}$, that specify a floating-point system. It is related to the *machine precision*,

which is defined as

$$\epsilon_M = 2^{-n}.$$

In practice (for IEEE double-precision numbers), $n = 53$ and

$$\epsilon_M = 2^{-53} \approx 1.1102 \cdot 10^{-16}.$$

The machine precision can be interpreted in several ways. We can first note that $1 + 2\epsilon_M$ is the smallest floating-point number greater than 1: the number 1 is represented as

$$1 = +(.100\cdots00)_2 \cdot 2^1,$$

so the next higher floating-point number is

$$+(.100\cdots01)_2 \cdot 2^1 = (2^{-1} + 2^{-n}) \cdot 2^1 = 1 + 2\epsilon_M.$$

More generally, $2\epsilon_M$ is the difference between consecutive floating-point numbers in the interval $[1, 2]$. In the interval $[2, 4]$, the spacing between floating-point numbers increases:

$$2 = +(.100\cdots00)_2 \cdot 2^2,$$

so the next number is

$$+(.100\cdots01)_2 \cdot 2^2 = (2^{-1} + 2^{-n}) \cdot 2^2 = 2 + 4\epsilon_M.$$

Similarly, the distance between consecutive numbers in $[4, 8]$ is $8\epsilon_M$, and so on.

17.3 Rounding

If we want to represent a number x that is not a floating-point number, we have to round it to the nearest floating-point number. We will denote by $\mathrm{fl}(x)$ the floating-point number closest to x, and refer to $\mathrm{fl}(x)$ as the floating-point representation of x. When there is a tie, *i.e.*, when x is exactly in the middle between two consecutive floating-point numbers, the floating-point number with least significant bit 0 (*i.e.*, $d_n = 0$) is chosen.

For example, as we have seen, the smallest floating-point number greater than 1 is $1 + 2\epsilon_M$. Therefore, $\mathrm{fl}(x) = 1$ for $1 \le x < 1 + \epsilon_M$, and $\mathrm{fl}(x) = 1 + 2\epsilon_M$ for $1 + \epsilon_M < x \le 1 + 2\epsilon_M$. The number $x = 1 + \epsilon_M$ is the midpoint of the interval $[1, 1 + 2\epsilon_M]$. In this case we choose $\mathrm{fl}(1 + \epsilon_M) = 1$, because

$$1 = +(.10\ldots00) \cdot 2^1, \qquad 1 + 2\epsilon_M = +(.10\ldots01) \cdot 2^1,$$

and the tie-breaking rule says that we choose the number with least significant bit 0. This provides a second interpretation of the machine precision: all numbers $x \in [1, 1 + \epsilon_M]$ are indistinguishable from 1.

The machine precision also provides a bound on the rounding error. For example, if $1 \le x \le 2$, the maximum rounding error is ϵ_M:

$$|\,\mathrm{fl}(x) - x| \le \epsilon_M$$

for $x \in [1, 2]$. For $x \in [2, 4]$, the maximum rounding error is $|\operatorname{fl}(x) - x| \leq 2\epsilon_M$, et cetera. It can also be shown that

$$\frac{|\operatorname{fl}(x) - x|}{x} \leq \epsilon_M$$

for all x. This bound gives a third interpretation of the machine precision: ϵ_M is an upper bound on the relative error that results from rounding a real number to the nearest floating-point number. In IEEE double precision arithmetic, this means that the relative error due to rounding is about $1.11 \cdot 10^{-16}$, *i.e.*, the precision is roughly equivalent to 15 or 16 significant digits in a decimal representation.

Exercises

17.1 Evaluate the following expressions in MATLAB and explain the results. MATLAB uses IEEE double precision arithmetic.

(a) `(1 + 1e-16) - 1`

(b) `1 + (1e-16 - 1)`

(c) `(1 - 1e-16) - 1`

(d) `(2 + 2e-16) - 2`

(e) `(2 - 2e-16) - 2`

(f) `(2 + 3e-16) - 2`

(g) `(2 - 3e-16) - 2`

17.2 Answer the following questions, assuming IEEE double precision arithmetic.

(a) What is the largest floating-point number less than $1/2$?

(b) What is the smallest floating-point number greater than 4 ?

(c) How many floating-point numbers are there in the interval $[1/2, 4)$?

17.3 How many IEEE double precision floating-point numbers are contained in the following intervals?

(a) The interval $[1/2, 3/2)$.

(b) The interval $[3/2, 5/2)$.

Explain your answer.

17.4 Run the following MATLAB code and explain the result.

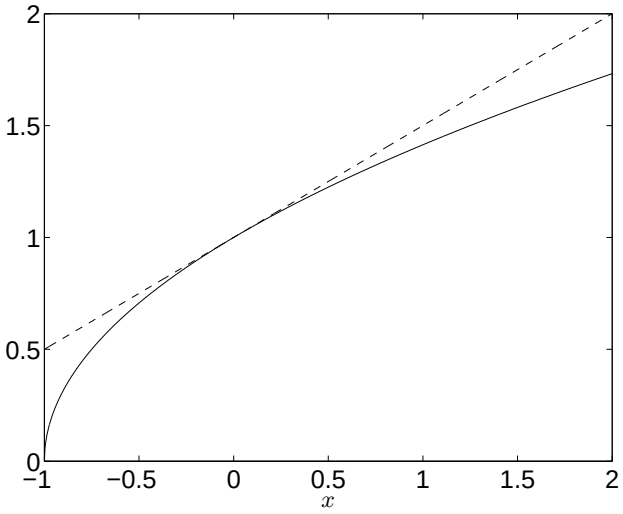
```
>> x = 2;
>> for i=1:54
     x = sqrt(x);
   end;
>> for i=1:54
     x = x^2;
   end;
>> x
```

Hint. If $x > 1$, then $1 < \sqrt{x} < 1 + \frac{1}{2}(x - 1)$.

17.5 Explain the following results in MATLAB. (Note that $\log(1 + x)/x \approx 1$ for small x so the correct result is very close to 1.)

```
>>  log(1+3e-16)/3e-16
ans =
    0.7401

>>  log(1+3e-16)/((1+3e-16)-1)
ans =
    1.0000
```

17.6 The figure shows $\sqrt{1 + x}$ around $x = 0$ (solid line) and its first-order Taylor approximation $1 + x/2$ (dashed line).



It is clear that $\sqrt{1+x} \approx 1 + (1/2)x$ for small x. Therefore the function

$$f(x) = \frac{\sqrt{1+x} - 1}{x}$$

is approximately equal to $1/2$ for x around zero. We evaluated f in MATLAB, using the command

```
y = (sqrt(1+x)-1)/x
```

and obtained the following results:

x	y
$2 \cdot 10^{-16}$	0
$3 \cdot 10^{-16}$	0
$4 \cdot 10^{-16}$	0.5551
$5 \cdot 10^{-16}$	0.4441

(a) Explain the four values of y.

(b) Give a more accurate method for evaluating $f(x)$ for values of x near 0.

17.7 One possible definition of the number $e = 2.7182818\ldots$ is as the limit

$$e = \lim_{n\to\infty} (1 + 1/n)^n.$$

This suggests a method for evaluating e: we pick a large n, and evaluate $(1+1/n)^n$. One would expect that this yields better approximations as n increases.
Evaluate $(1 + 1/n)^n$ in MATLAB for $n = 10^4$, $n = 10^8$, $n = 10^{12}$, and $n = 10^{16}$. How many correct digits do you obtain? Explain briefly.

17.8 The following lines of MATLAB code evaluate the function

$$f(x) = \frac{e^x - e^{-x}}{2x}$$

at $x = 1.5 \cdot 10^{-16}$ in four different ways.

(a) `x = 1.5e-16;  f = 0.5*((exp(x)-exp(-x))/x)`

(b) `x = 1.5e-16;  f = 0.5*((exp(x)-exp(-x))/((x+1)-1))`

(c) `x = 1.5e-16;  f = 0.5*((exp(x)-exp(-x))/((x-1)+1))`

(d) `x = 1.5e-16;  f = (exp(x)-exp(-x))/((x+1)+(x-1))`

What are the four computed values of `f`, if IEEE floating point arithmetic is used? You can assume that $\exp(x)$ and $\exp(-x)$ are calculated exactly and then rounded to the nearest floating-point number.

Hint: From Taylor's theorem,

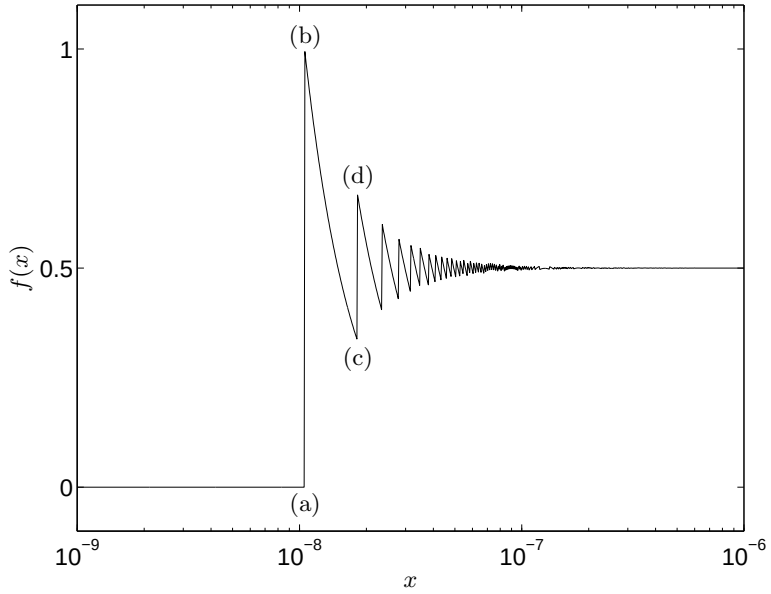
$$\exp(x) \;=\; 1 + x + \frac{1}{2}x^2 + \frac{1}{3!}x^3 + \cdots \;=\; 1 + x + \frac{\exp(\xi)}{2}x^2$$

for some ξ between 0 and x.

17.9 The plot shows the function

$$f(x) = \frac{1 - \cos(x)}{x^2},$$

evaluated in MATLAB using the command `(1-cos(x))/x^2`. We plot the function between $x = 10^{-9}$ and $x = 10^{-6}$, with a logarithmic scale for the x-axis.



We notice large errors: the correct value of $f(x)$ in this interval is very close to 1/2, because $\cos x \approx 1 - x^2/2$ for small x. We also note that the computed function is not continuous.

(a) At what value of x does the first discontinuity (from point (a) to (b)) occur? Why is the computed value zero to the left of point (a)?

(b) At what point does the second discontinuity occur (from point (c) to (d))?

(c) What are the computed values at points (c) and (d)?

(d) Give a more stable method for computing $f(x)$ for small x.

Be as precise as possible in your answers for parts a, b and c. However, you can use the approximation $\cos x \approx 1 - x^2/2$.

17.10 The inequality

$$\|a - b\| \geq \|a\| - \|b\|$$

holds for all vectors a and b of the same length. The MATLAB code below evaluates the difference of the two sides of the inequality for

$$a = \begin{bmatrix} 1 \\ 2 \cdot 10^{-8} \end{bmatrix}, \qquad b = \begin{bmatrix} 10^{-16} \\ 0 \end{bmatrix}.$$

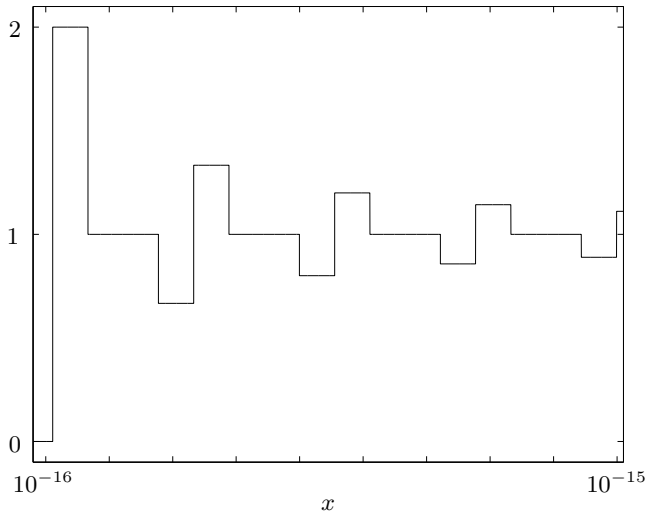
```
>> a = [1; 2e-8];
>> b = [1e-16; 0];
>> norm(a-b) - (norm(a) - norm(b))

ans =

  -2.2204e-16
```

The result is negative, which contradicts the inequality. Explain the number returned by MATLAB, assuming IEEE double precision arithmetic was used.
(You can use the linear approximation $\sqrt{1+x} \approx 1 + x/2$ for small x. The linear function $1 + x/2$ is also an upper bound on $\sqrt{1+x}$ for all $x \geq -1$.)

17.11 The figure shows the function

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

evaluated in IEEE double precision arithmetic in the interval $[10^{-16}, 10^{-15}]$, using the MATLAB command `((1+x)-1)/(1+(x-1))` to evaluate $f(x)$.



We notice that the computed function is piecewise-constant, instead of a constant 1.

(a) What are the endpoints of the intervals on which the computed values are constant?

(b) What are the computed values on each interval?

Carefully explain your answers.

17.12 The derivative of a function f at a point $\hat{x}$ can be approximated as

$$f'(\hat{x}) \approx \frac{f(\hat{x} + h) - f(\hat{x} - h)}{2h}$$

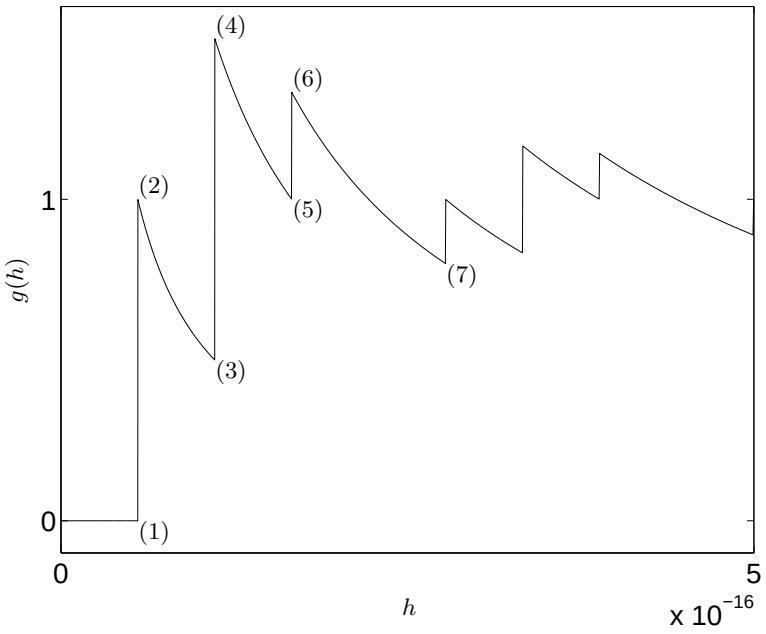
for small positive h. The right-hand side is known as a *finite-difference approximation* of the derivative.

The figure shows the finite-difference approximation of the derivative of $f(x) = \exp(x)$ at $\hat{x} = 0$, for values of h in the interval $(0, 5 \cdot 10^{-16}]$. The finite-difference approximation

$$g(h) = \frac{\exp(h) - \exp(-h)}{2h}$$

was computed using the MATLAB command

```
g = ( exp(h) - exp(-h) ) / ( 2*h )
```



Explain the first four segments of the graph, assuming IEEE double precision arithmetic was used in the calculation. You can make the approximation $\exp(t) \approx 1 + t$ for small t.

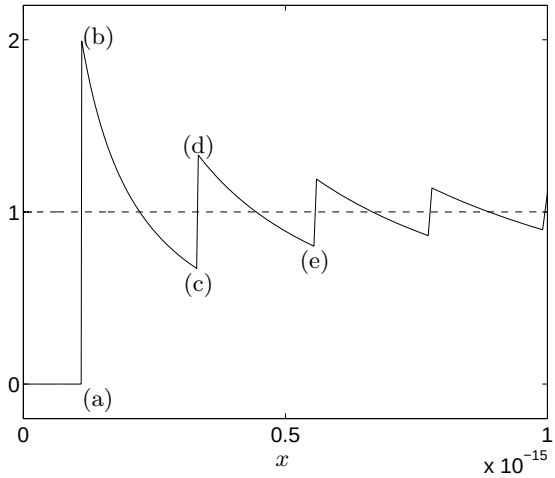
Your explanation should include the numerical values of the seven points marked on the graph, and an expression for the curve between points (2) and (3), (4) and (5), and (6) and (7).

17.13 The graphs in the figure are the two functions

$$f(x) = \frac{\exp(\log x)}{x}, \qquad g(x) = \frac{\log(\exp x)}{x},$$

evaluated on the interval $(0, 10^{-15}]$ with IEEE double precision arithmetic, using the MATLAB commands `exp(log(x))/x` and `log(exp(x))/x`. One of the graphs is shown in dashed line and the other in solid line.

Here, log denotes the natural logarithm, so the correct values are $f(x) = g(x) = 1$ for positive x. We see that the dashed line is quite accurate while the solid line is very inaccurate.



(a) Which of the two expressions (`exp(log(x))/x` or `log(exp(x))/x`) was used for the graph in solid line, and which one for the graph in dashed line? You can assume that the MATLAB functions `log(u)` and `exp(v)` return the exact values of $\log u$ and $\exp v$, rounded to the nearest floating-point number.

(b) Explain the graph in solid line up to point (e). What are (approximately) the horizontal and vertical values at the labeled points (a)–(e)? Why are the segments between (b) and (c), and between (d) and (e) nonlinear?

To analyze the effect of rounding error, you can use the first-order Taylor approximations $\log(a+b) \approx \log(a)+b/a$ for $a > 0$ and $|b| \ll a$, and $\exp(a+b) \approx \exp(a)(1+b)$ for $|b| \ll |a|$.