# Introduction to Matrix Methods and Applications

**(Working Title)**

S. Boyd and L. Vandenberghe

ROUGH DRAFT October 31, 2014

# Contents

# Chapter 1

# Vectors

In this chapter we introduce vectors and some common operations on them. We describe some settings in which vectors are used.

## 1.1 Vectors

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

$$\begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{bmatrix}.$$

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

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

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 above, for example, has size four; its third entry is 3.6. 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.

We often use symbols to denote vectors. If we denote an $n$-vector using the symbol $a$, the $i$th element of the vector $a$ is denoted $a_i$, where the subscript $i$ is an integer index that runs from 1 to $n$, the size of the vector.

Two vectors $a$ and $b$ are *equal*, which we denote $a = b$, if they have the same size, and each of the corresponding entries is the same. If $a$ and $b$ are $n$-vectors, then $a = b$ means $a_1 = b_1, \ldots, a_n = b_n$.

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 all real $n$-vectors is denoted $\mathbf{R}^n$, so $a \in \mathbf{R}^n$ is another way to say that $a$ is an $n$-vector with real entries. 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*.

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

$$a = \left[\begin{array}{c} b \\ c \\ d \end{array}\right],$$

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

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

The stacked vector $a$ is also written as $a = (b, c, d)$.

**Subvectors.** In the equation above, we say that $b$, $c$, and $d$ are *subvectors* or *slices* of $a$, with sizes $m$, $n$, and $p$, respectively. One notation used to denote subvectors uses *colon notation*. If $a$ is a vector, then $a_{r:s}$ is the vector of size $s-r+1$, with entries $a_r, \ldots, a_s$:

$$a_{r:s} = (a_r, \ldots, a_s).$$

The subscript $r\!:\!s$ is called the *index range*. Thus, in our example above, we have

$$b = a_{1:m}, \qquad c = a_{m+1:m+n}, \qquad d = a_{m+n+1:m+n+p}.$$

Colon notation is not completely standard, but is growing in popularity.

**Notational conventions.** Some authors try to use notation that helps the reader distinguish between vectors and scalars (numbers). For example, Greek letters ($\alpha$, $\beta$, ...) 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).

**Indexing.** 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 of length $n$ 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 $i$th element of a vector $a$. The same notation will occasionally refer to the $i$th vector in a collection or list of $k$ vectors $a_1, \ldots, a_k$. Whether $a_3$ means the third element of a vector $a$ (in which case $a_3$ is

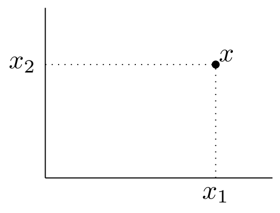

**Figure 1.1** The 2-vector $x$ specifies the position (shown as a dot) with coordinates $x_1$ and $x_2$ in a plane.

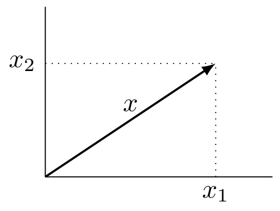

**Figure 1.2** The 2-vector $x$ represents a displacement in the plane (shown as an arrow) by $x_1$ in the first axis and $x_2$ in the second.

a number), or the third vector in some list of vectors (in which case $a_3$ is a vector) should be clear from the context. When we need to refer to an element of a vector that is in an indexed collection of vectors, we can write $(a_i)_j$ to refer to the $j$th entry of $a_i$, the $i$th vector in our list.

## 1.2 Examples

An $n$-vector can be used to represent $n$ quantities or values in an application. In some cases the values are similar in nature (for example, they are given in the same physical units); in others, the quantities represented by the entries of the vector are quite different from each other. We briefly describe below some typical examples, many of which we will see throughout the book.

**Location and displacement.** A 2-vector can be used to represent a position or location in a 2-dimensional space, *i.e.*, a plane, as shown in figure 1.1. A 3-vector is used to represent a location or position of some point in 3-dimensional space. The entries of the vector give the coordinates of the position or location.

A vector can also be used to represent a displacement in a plane or 3-dimensional space, in which case it is typically drawn as an arrow, as shown in figure 1.2. A vector can also be used to represent the velocity or acceleration, at a given time, of a point that moves in a plane or 3-dimensional space.

**Color.** A 3-vector can represent a color, with its entries giving the Red, Green, and Blue (RGB) intensity values (often between 0 and 1). The vector $(0, 0, 0)$ represents black, the vector $(0, 1, 0)$ represents a bright pure green color, and the vector $(1, 0.5, 0.5)$ represents a shade of pink.

**Quantities.** An $n$-vector $q$ can represent the amounts or quantities of $n$ different resources or products held (or produced, or required) by an entity such as a company. Negative entries mean an amount of the resource owed to another party (or consumed, or to be disposed of). For example, a *bill of materials* is a vector that gives the amounts of $n$ resources required to create a product or carry out a task.

**Portfolio.** An $n$-vector $h$ can represent a stock portfolio or investment in $n$ different assets, with $h_i$ giving the number of shares of asset $i$ held. The vector $(100, 50, 20)$ represents a portfolio consisting of 100 shares of asset 1, 50 shares of asset 2, and 20 shares of asset 3. Short positions (*i.e.*, shares that you owe another party) are represented by negative entries in a portfolio vector. (The entries of the portfolio vector can also be given in dollar values, or fractions of the total dollar amount invested.)

**Proportions.** A vector $w$ can be used to give fractions or proportions out of $n$ choices, outcomes, or options, with $w_i$ the fraction with choice or outcome $i$. In this case the entries are nonnegative and add up to one. Such vectors can also be interpreted as the recipes for a mixture of $n$ items, an allocation across $n$ entities, or as probability values in a probability space with $n$ outcomes. For example, a uniform mixture of 4 outcomes is represented as $(1/4, 1/4, 1/4, 1/4)$.

**Time series.** An $n$-vector can represent a *time series* or *signal*, that is, the value of some quantity at different times. (The entries in a vector that represents a time series are sometimes called *samples*, especially when the quantity is something measured.) An audio (sound) signal can be represented as a vector whose entries give the value of acoustic pressure at equally spaced times (typically 48000 or 44100 per second). A vector might give the hourly rainfall (or temperature, or pressure) at some location, over some time period.

**Daily return.** In finance, a vector can represent the daily return of a stock. In this example, the samples are not uniformly spaced in time; the index refers to trading days, and does not include weekends or market holidays. A vector can represent the daily (or quarterly, hourly, or minute-by-minute) value of any other quantity of interest for an asset, such as price, volume, or dividend.

**Cash flow.** A cash flow into and out of an entity (say, a company) can be represented by a vector (with positive representing payments to the entity, and negative representing payment by the entity). For example, with entries giving cash flow each quarter, the vector $(1000, -10, -10, -10, -1010)$ represents a one year loan of \$1000, with 1% interest only payments made each quarter, and the principal and last interest payment at the end.

**Images.** A monochrome (black and white) image of $M \times N$ pixels can be represented by a vector of length $MN$, with the elements giving grayscale levels at the pixel locations, typically ordered column-wise or row-wise. A color $M \times N$ pixel image would require a vector of length $3MN$, with the entries giving the R, G, and B values for each pixel, in some agreed-upon order.

**Video.** A monochrome video, *i.e.*, a time series of length $K$ images of $M \times N$ pixels, can be represented by a vector of length $KMN$ (again, in some particular order).

**Word counts and histogram.** A vector of length $n$ can represent the number of times each word in a dictionary of $n$ words appears in a document. For example, $(25, 2, 0)$ means that the first word appears 25 times, the second one twice, and the third one not at all. (Typical dictionaries have many more than 3 elements.) A variation is to have the entries of the vector give the *histogram* of word frequencies in the document, so that, *e.g.*, $x_5 = 0.003$ means that 0.3% of all the words in the document are the fifth word in the dictionary.

**Occurrence or subsets.** An $n$-vector $o$ can be used to record whether or not each of $n$ different events has occurred, with $o_i = 0$ meaning that event $i$ did not occur, and $o_i = 1$ meaning that it did occur. Such a vector encodes a subset of a set of $n$ objects, with $o_i = 1$ meaning that object $i$ is contained in the subset, and $o_i = 0$ meaning that object $i$ is not contained.

**Features or attributes.** In many applications a vector collects a set of $n$ quantities that pertain to a single thing or object. The quantities can be measurements, or quantities that can be measured or derived from the object. For example, a 6-vector $f$ could give the age, height, weight, blood pressure, temperature, and gender of a patient admitted to a hospital. (The last entry of the vector could be encoded as $f_6 = 0$ for Male, $f_6 = 1$ for Female.) In this example, the quantities represented by the entries of the vector are quite different, with different physical units.

## 1.3 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 computer 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 $i$th unit vector (of size $n$) is the unit vector with $i$th element one, and is denoted $e_i$. For example, 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 above. Here, $e_i$ denotes the $i$th unit vector, and not the $i$th element of a vector $e$. Thus we can describe the $i$th unit $n$-vector $e_i$ as

$$(e_i)_j = \begin{cases} 1 & j = i \\ 0 & j \neq i, \end{cases}$$

for $i, j = 1, \ldots, n$. On the left-hand side $e_i$ is an $n$-vector; $(e_i)_j$ is a number, its $j$th entry. 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.)

**Sparsity.** A vector is said to be *sparse* if many of its entries are zero; its *sparsity pattern* is the set of indices of nonzero entries. The number of the nonzero entries of an $n$-vector $x$ is denoted $\mathbf{nnz}(x)$.

Sparse vectors arise in many applications. On a computer, they can be efficiently stored, and various operations on them can be carried out more efficiently. Unit vectors are sparse, since they have only one nonzero entry. The zero vector is the sparsest possible vector, since it has no nonzero entries.

## 1.4 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, called the *sum* of the vectors. 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 result of vector subtraction is called the *difference* of the two vectors.

Several properties of vector addition are easily verified. For any vectors $a$, $b$, and $c$ of the same size we have the following.

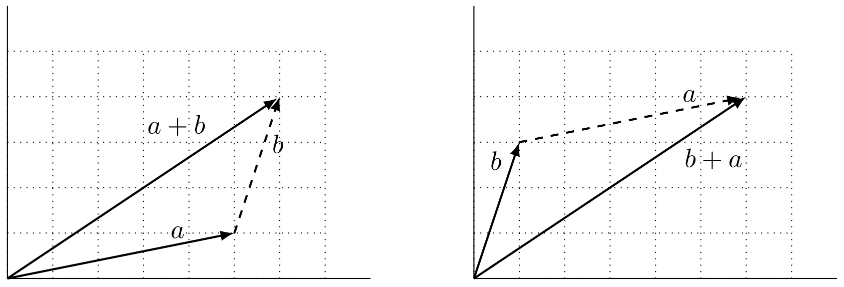

**Figure 1.3** *Left.* The lowest dark arrow shows the displacement $a$; the displacement $b$, shown as a dashed arrow, starts from the head of the displacement $a$ and ends at the sum displacement $a+b$, shown as the longer dark arrow. *Right.* The displacement $b+a$.

- Vector addition is *commutative*: $a+b=b+a$.
- 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 $a$.)
- $a-a=0$. Subtracting a vector from itself yields the zero vector.

Some computer languages for manipulating vectors define 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.

**Examples.**

- *Displacements.* When vectors $a$ and $b$ represent displacements, the sum $a+b$ is the net displacement found by first displacing by $a$, then displacing by $b$, as shown in figure 1.3. Note that we arrive at the same vector if we first displace by $b$ and then $a$. If the vector $p$ represents a position and the vector $a$ represents a displacement, then $p+a$ is the position of the point $p$, displaced by $a$, as shown in figure 1.4.
- *Displacements between two points.* If the vectors $p$ and $q$ represent the positions of two points in 2- or 3-dimensional space, then $p-q$ is the displacement vector from $q$ to $p$, as illustrated in figure 1.5.
- *Word counts.* If $a$ and $b$ are word count vectors (using the same dictionary) for two documents, the sum $a+b$ is the word count vector of a new document created by combining the original two (in either order). The word count difference vector $a-b$ gives the number of times more each word appears in the first document than the second.

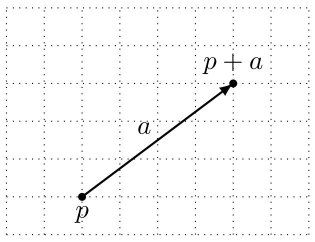

**Figure 1.4** The vector $p+a$ is the position of the point represented by $p$ displaced by the displacement represented by $a$.

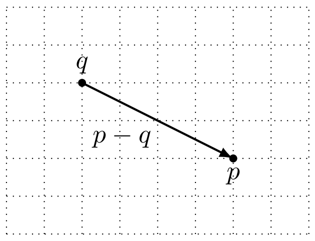

**Figure 1.5** The vector $p-q$ represents the displacement from the point represented by $q$ to the point represented by $p$.

- *Bill of materials.* Suppose $q_1, \ldots, q_N$ are $n$-vectors that give the quantities of $n$ different resources required to accomplish $N$ tasks. Then the sum $n$-vector $q_1 + \cdots + q_N$ gives the bill of materials for completing all $N$ tasks.
- *Market clearing.* Suppose the $n$-vector $q_i$ represents the amounts of $n$ goods or resources produced (when positive) or consumed (when negative) by agent $i$, for $i = 1, \ldots, N$, so $(q_5)_4 = -3.2$ means that agent 5 consumes 3.2 units of resource 4. The sum $s = q_1 + \cdots + q_N$ is the $n$-vector of total net surplus of the resources (or shortfall, when the entries are negative). When $s = 0$, we have a closed market, which means that the total amount of each resource produced by the agents balances the total amount consumed. In other words, the $n$ resources are *exchanged* among the agents. In this case we say that the *market clears* (with the resource vectors $q_1, \ldots, q_N$).
- *Audio addition.* When $a$ and $b$ are vectors representing audio signals over the same period of time, the sum $a+b$ is an audio signal that is perceived as containing both audio signals combined into one. If $a$ represents a recording of a voice, and $b$ a recording of music (of the same length), the audio signal $a+b$ will be perceived as containing both the voice recording and, simultaneously, the music.
- *Feature differences.* If $f$ and $g$ are $n$-vectors that give $n$ feature values for two items, the difference vector $d = f - g$ gives the difference in feature values for the two objects. For example, $d_7 = 0$ means that the two objects have the

same value for feature 7; $d_3 = 1.67$ means that the first object's third feature value exceeds the second object's third feature value by 1.67.

## 1.5 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, typically with the scalar on the left, as in

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

Scalar-vector multiplication can also be written with the scalar on the right, as in

$$\begin{bmatrix} 1 \\ 9 \\ 6 \end{bmatrix}(1.5) = \begin{bmatrix} 1.5 \\ 13.5 \\ 9 \end{bmatrix}.$$

The meaning is the same: it is the vector obtained by multiplying each element by the scalar. (A similar notation you might see is $a/2$, where $a$ is a vector, meaning $(1/2)a$.) The scalar-vector 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 $a$).

By definition, we have $\alpha a = a\alpha$, for any scalar $\alpha$ and any vector $a$. This is called the *commutative property* of scalar-vector multiplication; it means that scalar-vector multiplication can be written in either order.

Scalar multiplication obeys several other laws that are easy to figure out from the definition. For example, it satisfies the associative property: If $a$ is a vector and $\beta$ and $\gamma$ are scalars, we have

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

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. As a consequence, we can write the vector above as $\beta\gamma a$, since it does not matter whether we interpret this as $\beta(\gamma a)$ or $(\beta\gamma)a$.

The associative property holds also when we denote scalar-vector multiplication with the scalar on the right. For example, we have $\beta(\gamma a) = (\beta a)\gamma$, and consequently we can write both as $\beta a\gamma$. As a convention, however, this vector is normally written as $\beta\gamma a$ or as $(\beta\gamma)a$.

If $a$ is a vector and $\beta$, $\gamma$ are scalars, then

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

(This is the left-distributive property of scalar-vector multiplication.) Scalar multiplication, like ordinary multiplication, has higher precedence in equations 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. When scalar multiplication is written with the scalar on the right, we have the right-distributive property. Scalar-vector multiplication also satisfies the right-distributive property:

$$\beta(a + b) = \beta a + \beta b$$

for any scalar $\beta$ and any $n$-vectors $a$ and $b$.

**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 $n$-vector $b$ as a linear combination of the standard unit vectors, as

$$b = b_1 e_1 + \cdots + b_n e_n. \qquad (1.1)$$

In this equation $b_i$ are the entries in $b$ (*i.e.*, scalars), and $e_i$ is the $i$th unit vector. A specific example is

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

**Special linear combinations.** Some linear combinations of the vectors $a_1, \ldots, a_m$ have special names. For example, the linear combination with $\beta_1 = \cdots = \beta_m = 1$, given by $a_1 + \cdots + a_m$, is the *sum* of the vectors, and the linear combination with $\beta_1 = \cdots = \beta_m = 1/m$, given by $(1/m)(a_1 + \cdots + a_m)$, is the *average* of the vectors. When the coefficients sum to one, *i.e.*, $\beta_1 + \cdots + \beta_m = 1$, the linear combination is called an *affine combination.* When the coefficients in an affine combination are nonnegative, it is called a *convex combination* or a *mixture.* The coefficients in an affine or convex combination are sometimes given as percentages, which add up to 100%.

**Linear combinations of linear combinations.** If vectors $b_1, \ldots, b_k$ are each a linear combination of vectors $a_1, \ldots, a_m$, and $c$ is a linear combination of $b_1, \ldots, b_k$, then $c$ is a linear combination of $a_1, \ldots, a_m$. We will encounter this important idea in several later chapters, and will introduce notation to describe it concisely then.

**Examples.**

- *Displacements.* When a vector $a$ represents a displacement, and $\beta > 0$, $\beta a$ is a displacement in the same direction of $a$, with its magnitude scaled by $\beta$. When $\beta < 0$, $\beta a$ represents a displacement in the opposite direction of $a$, with magnitude scaled by $|\beta|$. This is illustrated in figure 1.6.

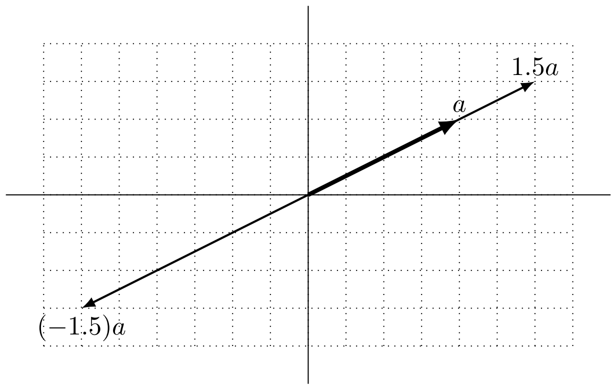

**Figure 1.6** $1.5a$ represents the displacement in the direction of the displacement $a$, with magnitude scaled by 1.5; $(-1.5)a$ represents the displacement in the opposite direction, also with magnitude scaled by 1.5.

- *Audio scaling.* If $a$ is a vector representing an audio signal, the scalar-vector product $\beta a$ is perceived as the same audio signal, but changed in volume (loudness) by the factor $|\beta|$. For example, when $\beta = 1/2$ (or $\beta = -1/2$), $\beta a$ is perceived as the same audio signal, but quieter.
- *Audio mixing.* When $a_1, \ldots, a_m$ are vectors representing audio signals (over the same period of time, for example, simultaneously recorded), they are called *tracks*. The linear combination $\beta_1 a_1 + \cdots + \beta_m a_m$ is perceived as a mixture (also called a *mix*) of the audio tracks, with relative loudness given by $|\beta_1|, \ldots, |\beta_m|$. A producer in a studio, or a sound engineer at a live show, chooses values of $\beta_1, \ldots, \beta_m$ to give a good balance between the different instruments, vocals, and drums.

## 1.6 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 sum of the products of corresponding entries. (The origin of the superscript in the inner product notation $a^T b$ will be explained in chapter 4.) 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 §6.1. As a specific example of the inner product, we have

$$\begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix} = -7.$$

When $n = 1$, the inner product reduces to the usual product of two numbers.

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 $\gamma$ is a scalar, we have the following.

- *Commutativity.* $a^Tb = b^Ta$. The order of the two vector arguments in the inner product does not matter.
- *Associativity with scalar multiplication.* $(\gamma a)^Tb = \gamma(a^Tb)$, so we can write both as $\gamma a^Tb$.
- *Disitributivity with vector addition.* $(a+b)^Tc = a^Tc+b^Tc$. The inner product can be distributed across vector addition.

These can be combined to obtain other identities, such as $a^T(\gamma b) = \gamma(a^Tb)$, or $a^T(b + \gamma c) = a^Tb + \gamma a^Tc$. As another useful example, we have, for any vectors $a, b, c, d$ of the same size,

$$(a + b)^T(c + d) = a^Tc + a^Td + b^Tc + b^Td.$$

This formula expresses an inner product on the left-hand side as a sum of four inner products on the right-hand side, and is analogous to expanding a product of sums in algebra. Note that on the left-hand side, the two addition symbols refer to vector addition, whereas on the right-hand side, the three addition symbols refer to scalar (number) addition.

**General examples.**

- *Unit vector.* $e_i^Ta = a_i$. The inner product of a vector with the $i$th standard unit vector gives (or 'picks out') the $i$th element $a$.
- *Sum.* $\mathbf{1}^Ta = a_1 + \cdots + a_n$. The inner product of a vector with the vector of ones gives the sum of the elements of the vector.
- *Average.* $((1/n)\mathbf{1})^Ta = (a_1 + \cdots + a_n)/n$. The inner product of an $n$-vector with $(1/n)\mathbf{1}$ gives the average of the elements of the vector.
- *Sum of squares.* $a^Ta = a_1^2 + \cdots + a_n^2$. The inner product of a vector with itself gives the sum of the squares of the elements of the vector.
- *Selective sum.* Let $b$ be a vector all of whose entries are either 0 or 1. Then $b^Ta$ is the sum of the elements in $a$ for which $b_i = 1$.

**Block vectors.** If the vectors $a$ and $b$ are block vectors, and the corresponding blocks have the same sizes (in which case we say they *conform*), then we have

$$a^Tb = \begin{bmatrix} a_1 \\ \vdots \\ a_k \end{bmatrix}^T \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix} = a_1^Tb_1 + \cdots + a_k^Tb_k.$$

The inner product of block vectors is the sum of the inner products of the blocks.

**Applications.** The inner product is useful in many applications, a few of which we list here.

- *Co-occurrence.* If $a$ and $b$ are $n$-vectors that describe occurrence, *i.e.*, each of their elements is either 0 or 1, then $a^T b$ gives the total number of indices for which $a_i$ and $b_i$ are both one, that is, the total number of co-occurrences. If we interpret the vectors $a$ and $b$ as describing subsets of $n$ objects, then $a^T b$ gives the number of objects in the intersection of the two subsets.

- *Weights, features, and score.* When the vector $f$ represents a set of features of an object, and $w$ is a vector of same size (often called a *weight vector*), the inner product $w^T f$ is the sum of the feature values, weighted by the weights, and is sometimes called a *score*. For example, if the features are associated with a loan applicant (*e.g.*, age, income, ...) we might interpret $s = w^T f$ as a credit score. In this example we can interpret $w_i$ as the weight given to feature $i$ in forming the score.

- *Price-quantity.* If $p$ represents a vector of prices of $n$ goods, and $q$ is a vector of quantities of the $n$ goods (say, the bill of materials for a product), then their inner product $p^T q$ is the total cost of the goods given by the vector $q$.

- *Speed-time.* Suppose the vector $s$ gives the speed of a vehicle traveling over $n$ segments, with $t$ a vector of the time taken to traverse the segments. Then $s^T t$ is the total distance travelled.

- *Probability and expected values.* Suppose the $n$-vector $p$ has nonnegative entries that sum to one, so it describes a set of proportions among $n$ items, or a set of probabilities of $n$ outcomes, one of which must occur. Suppose $f$ is another $n$-vector, where we interpret $f_i$ as the value of some quantity if outcome $i$ occurs. Then $f^T p$ gives the expected value or average of the quantity, under the probabilities (or fractions) given by $p$.

- *Polynomial evaluation.* Suppose the $n$-vector $c$ represents the coefficients of a polynomial $p$ of degree $n-1$ or less:

$$p(x) = c_n x^{n-1} + c_{n-1} x^{n-2} + \cdots + c_2 x + c_1.$$

  Let $t$ be a number, and let $z = (1, t, t^2, \ldots, t^{n-1})$ be the $n$-vector of powers of $t$. Then $c^T z = p(t)$, the value of the polynomial $p$ at the point $t$. So the inner product of a polynomial coefficient vector and vector of powers of a number evaluates the polynomial at the number.

- *Discounted total.* Let $c$ be an $n$-vector representing a cash flow, with $c_i$ be the cash received (when $c_i > 0$) in period $i$. Let $d$ be the $n$-vector defined as

$$d = (1, 1/(1+r), \ldots, 1/(1+r)^{n-1}),$$

  where $r > 0$ is an interest rate. Then

$$d^T c = c_1 + c_2/(1+r) + \cdots + c_n/(1+r)^{n-1}$$

  is the discounted total of the cash flow, *i.e.*, its *net present value* (NPV), with interest rate $r$.

- *Portfolio value.* Suppose $h$ is an $n$-vector representing the holdings in a portfolio of $n$ different assets (say, in shares, with negative meaning short positions). If $p$ is an $n$-vector giving the prices of the assets, then $p^T h$ is the total value of the portfolio.

- *Portfolio return.* Suppose $r$ is the vector of (fractional) returns of $n$ assets over some time period, *i.e.*, the asset relative price changes

$$r_i = \frac{p_i^{\text{final}} - p_i^{\text{initial}}}{p_i^{\text{initial}}}, \quad i = 1, \ldots, n,$$

where $p_i^{\text{initial}}$ and $p_i^{\text{final}}$ are the prices of asset $i$ at the beginning and end of the investment period. If $h$ is an $n$-vector giving our portfolio, with $h_i$ denoting the dollar value of asset $i$ held, then the inner product $r^T h$ is the total return of the portfolio, in dollars, over the period. If $w$ represents the fractional (dollar) amounts of our portfolio, then $r^T w$ gives the total return of the portfolio. For example, if $r^T w = 0.09$, then our portfolio return is 9%. If we had invested \$10000 initially, we would have earned \$900.

## 1.7 Linear functions

**Function notation.** 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 $f(x)$, which is a scalar, denotes the value of the function $f$ at $x$. (In the notation $f(x)$, $x$ is referred to as the *argument* of the function.) We can also interpret $f$ as a function of $n$ scalar arguments, the entries of the vector argument, in which case we write $f(x)$ as

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

Here we refer to $x_1, \ldots, x_n$ as the arguments of $f$.

To describe a function $f : \mathbf{R}^n \to \mathbf{R}$, we have to specify what its value is for any possible argument $x \in \mathbf{R}^n$. For example, we can define a function $f : \mathbf{R}^4 \to \mathbf{R}$ by

$$f(x) = x_1 + x_2 - x_4^2$$

for any 4-vector $x$. In words, we might describe $f$ as the sum of the first two elements of its argument, minus the square of the last entry of the argument. (This particular function does not depend on the third element of its argument.)

Sometimes we introduce a function without formally assigning a symbol for it, by directly giving a formula for its value in terms of its arguments, or describing how to find its value from its arguments. An example is the *sum function*, whose value is $x_1 + \cdots + x_n$. We can give a name to the value of the function, as in $y = x_1 + \cdots + x_n$, and say that $y$ is a function of $x$, in this case, the sum of its entries.

**The inner product function.** 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 used in forming the weighted sum.

**Superposition and linearity.** The inner product 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 $\alpha$, $\beta$. This property is called *superposition*. A function that satisfies the superposition property is called *linear*. We have just shown 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*.

If a function $f$ is linear, superposition extends to linear combinations of any number of vectors, and not just linear combinations of two vectors: We have

$$f(\alpha_1 x_1 + \cdots + \alpha_k x_k) = \alpha_1 f(x_1) + \cdots + \alpha_k f(x_k),$$

for any $n$ vectors $x_1, \ldots, x_k$, and any scalars $\alpha_1, \ldots, \alpha_k$. (This more general $k$-term form of superposition reduces to the two-term form given above when $k = 2$.) To see this, we note that

$$\begin{aligned} f(\alpha_1 x_1 + \cdots + \alpha_k x_k) &= \alpha_1 f(x_1) + f(\alpha_2 x_2 + \cdots + \alpha_k x_k) \\ &= \alpha_1 f(x_1) + \alpha_2 f(x_2) + f(\alpha_3 x_3 + \cdots + \alpha_k x_k) \\ &\;\;\vdots \\ &= \alpha_1 f(x_1) + \cdots + \alpha_k f(x_k). \end{aligned}$$

In the first line here, we apply (two-term) superposition to the argument

$$\alpha_1 x_1 + (1)(\alpha_2 x_2 + \cdots + \alpha_k x_k),$$

and in the other lines we apply this recursively.

**Inner product representation of a linear function.** We saw above that a function defined as the inner product of its argument with some fixed vector is linear. The converse is also true: If a function is linear, then it can be expressed as the inner product of its argument with some fixed vector.

Suppose $f$ is a scalar valued function of $n$-vectors, and is linear, *i.e.*, (1.3) holds for all $n$-vectors $x$, $y$, and all scalars $\alpha$, $\beta$. 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 by multi-term superposition we have

$$\begin{array}{rcl} f(x) & = & f(x_1 e_1 + \cdots + x_n e_n) \\ & = & x_1 f(e_1) + \cdots + x_n f(e_n) \\ & = & a^T x, \end{array}$$

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

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

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*, which means that 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 *mean* or *average* value of an $n$-vector is defined as

  $$f(x) = (x_1 + x_2 + \cdots + x_n)/n,$$

  and is denoted $\mathbf{avg}(x)$ (and sometimes $\overline{x}$). The average of a vector is a linear function. It can be expressed as $\mathbf{avg}(x) = a^T x$ with

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

- *Maximum.* The maximum element of an $n$-vector $x$, $f(x) = \max\{x_1, \ldots, x_n\}$, is not a linear function (except when $n = 1$). We can show this by a counterexample for $n = 2$. Take $x = (1, -1)$, $y = (-1, 1)$, $\alpha = 1$, $\beta = 1$. 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 $b$, which is sometimes called the *offset*. 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 $\alpha$, $\beta$ that satisfy $\alpha + \beta = 1$. For linear functions, superposition holds for *any* coefficients $\alpha$ and $\beta$; for affine functions, it holds *when the coefficients sum to one*, (*i.e.*, when the argument is an affine combination). 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

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

which holds when $f$ is affine, and $x$ is any $n$-vector (see exercise **??**). 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, when $x$ is a scalar, the function $f$ defined as $f(x) = \alpha x + \beta$ is sometimes referred to as a linear function of $x$, perhaps because its graph is a line. But when $\beta \neq 0$, $f$ is not a linear function of $x$, in the standard mathematical sense; it *is* an affine function of $x$. In this book we will distinguish between linear and affine functions.

## 1.8 Approximate linear models

In many applications, scalar-valued functions of $n$ variables, or relations between $n$ variables and a scalar one, can be *approximated* as linear or affine functions. In these cases we sometimes refer to the linear or affine function relating the set of variables and the scalar variable as a *model*, to remind us that the relation is only an approximation, and not exact. We give a few examples here.

### 1.8.1 First-order Taylor approximation

Suppose that $f : \mathbf{R}^n \to \mathbf{R}$ is differentiable, and $z$ is an $n$-vector. The *first-order Taylor approximation* of $f$ near (or at) the point $z$ is the function $\hat{f}(x)$ of $x$ defined as

$$\hat{f}(x) = f(z) + \frac{\partial f}{\partial x_1}(z)(x_1 - z_1) + \cdots + \frac{\partial f}{\partial x_n}(z)(x_n - z_n),$$

where $\frac{\partial f}{\partial x_i}(z)$ denotes the partial derivative of $f$ with respect to its $i$th argument, evaluated at the $n$-vector $z$. The hat appearing over $f$ on the left-hand side is a common notational hint that it is an approximation of the function $f$. The first-order Taylor approximation $\hat{f}(x)$ is a very good approximation of $f(x)$ when $x_i$ are near $z_i$. Sometimes $\hat{f}$ is written with a second vector argument, as $\hat{f}(x; z)$, to show the point $z$ at which the approximation is developed. The first term in the Taylor approximation is a constant; the other terms can be interpreted as the contribution to the (approximate) change in the function value (from $f(z)$) due to the change in $x_i$ (from $z_i$).

Evidently $\hat{f}$ is an affine function of $x$. (It is sometimes called the *linear approximation* of $f$ near $z$, even though it is in general affine, and not linear.) It can be written compactly using inner product notation as

$$\hat{f}(x) = f(z) + \nabla f(z)^T (x - z), \tag{1.5}$$

where $\nabla f(z)$ is an $n$-vector, the *gradient of* $f$ (at the point $z$),

$$\nabla f(z) = \left[ \begin{array}{c} \frac{\partial f}{\partial x_1}(z) \\ \vdots \\ \frac{\partial f}{\partial x_n}(z) \end{array} \right]. \tag{1.6}$$

We can express the first-order Taylor approximation as a linear function plus a constant,

$$\hat{f}(x) = \nabla f(z)^T x + (f(z) - \nabla f(z)^T z),$$

but the form (1.5) is perhaps easier to interpret.

The first-order Taylor approximation gives us an organized way to construct an affine approximation of a function $f : \mathbf{R}^n \to \mathbf{R}$, near a given point $z$, when there is a formula or equation that describes $f$, and it is differentiable.

### 1.8.2 Regression model

Suppose the $n$-vector $x$ represents a feature vector. The affine function of $x$ given by

$$y = x^T \beta + v, \tag{1.7}$$

where $\beta$ is an $n$-vector and $v$ is a scalar, is called a *regression model*. In this context, the entries of $x$ are called the *regressors*, and $y$ is called the *dependent variable*, *outcome*, or *label*. The vector $\beta$ is called the *weight vector*, and the scalar $v$ is called the *offset* in the regression model. Together, $\beta$ and $v$ are called the *parameters* in the regression model. We will see in chapter 8 how the parameters in a regression model can be estimated or guessed, based on some past or known observations of the feature vector $x$ and the outcome $y$. The symbol $\hat{y}$ is often used in place of $y$ in the regression model, to emphasize that it is an *estimate* or *prediction* of some outcome.

The entries in the weight vector have a simple interpretation: $\beta_i$ is the amount by which $y$ increases (if $\beta_i > 0$) when feature $i$ increases by one (with all other features the same). If $\beta_i$ is small, the outcome $y$ doesn't depend too strongly on feature $i$. The offset $v$ is the value of $y$ when all features have the value 0.

Vector stacking can be used to lump the weights and offset in the regression model (1.7) into a single parameter vector, which simplifies the notation a bit. We create a new regression vector $\tilde{x}$, with $n+1$ entries, as $\tilde{x} = (1, x)$. We can think of $\tilde{x}$ as a new feature vector, consisting of all $n$ original features, and one new feature added ($\tilde{x}_1$) at the beginning, which always has the value one. We define the parameter vector $\tilde{\beta} = (v, \beta)$, so the regression model (1.7) has the simple inner product form

$$y = x^T\beta + v = \left[\begin{array}{c} 1 \\ x \end{array}\right]^T \left[\begin{array}{c} v \\ \beta \end{array}\right] = \tilde{x}^T\tilde{\beta}. \qquad (1.8)$$

Often we omit the tildes, and simply write this as $y = x^T\beta$, where we assume that the first feature in $x$ is the constant 1. A feature that always has the value 1 is not particularly informative or interesting, but it does simplify the notation.

**Example.** As a simple example of a regression model, suppose that $y$ is the selling price of a house in some neighborhood, and $x$ contains some attributes of the house:

- $x_1$ is the lot size (in acres),
- $x_2$ is the house area (in 1000 square feet),
- $x_3$ is the number of bedrooms, and
- $x_4$ is the number of bathrooms.

If $y$ represents the selling price of the house, in thousands of dollars, the regression model $y = x^T\beta + v$ expresses the price in terms of the attributes or features. The individual weights $\beta_i$ are readily interpreted: $\beta_3$, for example, tells us how much more a house sells for with the addition of one bedroom. This regression model is not meant to describe an exact relationship between the house attributes and its selling price; it is a model or approximation.

## 1.9 Complexity of vector computations

**Flop counts and complexity.** Current computers can readily store and manipulate vectors with sizes measured in millions or even billions. So far we have seen only a few vector operations, like scalar multiplication, vector addition, and the inner product. How quickly these operations can be carried out depends very much on the computer hardware and software, and the size of the vector.

A very rough estimate of the time required to carry out some computation, such as an inner product, can be found by counting the total number of basic arithmetic operations (addition, subtraction, multiplication, and division of two numbers). Numbers are stored in *floating point format* on computers, so these operations are called *floating point operations*, or FLOPS. This term is in such common use that the acronym is now written in lower case letters, as flops, and the speed with which a computer can carry out flops is expressed in Gflops (billions of flops per second). Typical current values are in the range of 1–10 Gflop, but this can vary by several orders of magnitude. The actual time it takes a computer to carry out some computation depends on many other factors beyond the total number of flops required, so time estimates based on counting flops are very crude, and are not meant to be more accurate than a factor of ten or so. For this reason, gross approximations (such as ignoring a factor of 2) can be used when counting the flops required in a computation.

The *complexity* of an operation is the number of flops required to carry it out, as a function of the size or sizes of the input to the operation. Usually the complexity is highly simplified, dropping terms that are small or negligible when the sizes of the inputs are large.

**Complexity of vector operations.** Scalar-vector multiplication $ax$, where $x$ is an $n$-vector, requires $n$ multiplications, *i.e.*, $ax_i$ for $i = 1, \ldots, n$. Vector addition $x+y$ of two $n$-vectors takes $n$ additions, *i.e.*, $x_i + y_i$ for $i = 1, \ldots, n$. Computing the inner product $x^T y = x_1 y_1 + \cdots + x_n y_n$ of two $n$-vectors takes $2n-1$ flops, $n$ scalar multiplications and $n-1$ scalar additions. So scalar multiplication, vector addition, and the inner product of $n$-vectors require $n$, $n$, and $2n-1$ flops, respectively. We only need an estimate, so we simplify the last to $2n$ flops, and say that the *complexity* of scalar multiplication, vector addition, and the inner product of $n$-vectors is $n$, $n$, and $2n$ flops, respectively.

We can guess that a 1 Gflop computer can compute the inner product of two vectors of size one million in around one thousandth of a second, but we should not be surprised if the actual time differs by a factor of 10 from this value.

The *order* of the computation is obtained by ignoring any constant that multiplies a power of the dimension. So we say that the three vector operations scalar multiplication, vector addition, and inner product have order $n$. Ignoring the factor of 2 dropped in the actual complexity of the inner product is reasonable, since we do not expect flop counts to predict the running time with an accuracy better than a factor of 2. The order is useful in understanding how the time to execute the computation will scale when the size of its arguments changes. An order $n$ computation should take around 10 times longer to carry out its computation on an input that is 10 times bigger.

**Complexity of sparse vector operations.** If $x$ is sparse, then computing $ax$ requires $\mathbf{nnz}(x)$ flops. If $x$ and $y$ are sparse, computing $x+y$ requires no more than $\min\{\mathbf{nnz}(x), \mathbf{nnz}(y)\}$ flops (since no arithmetic operations are required to compute $(x+y)_i$ when either $x_i$ or $y_i$ is zero). If the sparsity patterns of $x$ and $y$ do not overlap (intersect), then zero flops are needed to compute $x+y$. The inner product calculation is similar: computing $x^T y$ requires no more than $2\min\{\mathbf{nnz}(x), \mathbf{nnz}(y)\}$ flops. When the sparisty patterns of $x$ and $y$ do not overlap, computing $x^T y$ requires zero flops, since $x^T y = 0$ in this case.

# Chapter 2

# Norm and distance

In this chapter we focus on the norm of a vector, a measure of its length, and on related concepts like distance, angle, and standard deviation. We finish with the $k$-means algorithm, which is used to cluster or partition a set of vectors into groups.

## 2.1 Norm

The *Euclidean norm* of a vector $x$, denoted $\|x\|$, is the squareroot of the sum of the squares of 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 $\|x\|_2$. Other less widely used terms for the Euclidean norm of a vector are the *magnitude*, or *length*, of a vector. We can express the Euclidean norm in terms of the inner product of $x$ with itself:

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

We use the same notation for the norm of vectors of different dimensions.

As simple examples, we have

$$\left\| \begin{bmatrix} 2 \\ -1 \\ 2 \end{bmatrix} \right\| = \sqrt{9} = 3, \qquad \left\| \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right\| = 1.$$

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. Like the absolute value of a number, the norm of a vector is a measure of its size. We say a vector is *small* if its norm is a small number, and we say it is *large* if its norm is a large number. (The numerical values of the norm that qualify for small or large depends on the particular application and context.)

Some important properties of the Euclidean norm are given below. Here $x$ and $y$ are vectors of the same size, and $\beta$ is a scalar.

- *Homogeneity.* $\|\beta x\| = |\beta| \|x\|$. Multiplying a vector by a scalar multiplies the norm by the absolute value of the scalar.
- *Triangle inequality.* $\|x + y\| \leq \|x\| + \|y\|$. The Euclidean norm of a sum of two vectors is no more than the sum of their norms. (The name of this property will be explained later.)
- *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. For example, let's verify the definiteness property. If $\|x\| = 0$, then we also have $\|x\|^2 = 0$, which means that $x_1^2 + \cdots + x_n^2 = 0$. This is a sum of $n$ nonnegative numbers, which is zero. We can conclude that each of the $n$ numbers is zero, since if any of them were nonzero the sum would be positive. So we conclude that $x_i^2 = 0$ for $i = 1, \ldots, n$, and therefore $x_i = 0$, $i = 1, \ldots, n$; and thus, $x = 0$. Establishing the second property, the triangle inequality, is not as easy; we will give a derivation a bit later.

Any real-valued function of an $n$-vector that satisfies the four properties listed above is called a (general) norm. But in this book we will only use the Euclidean norm, so from now on, we refer to the Euclidean norm as the norm. (See exercise **??**, which describes some other useful norms.)

**Root-mean-square value.** The norm is related to the *root-mean-square* (RMS) value of an $n$-vector $x$, defined as

$$\mathbf{rms}(x) = \sqrt{\frac{x_1^2 + \cdots + x_n^2}{n}} = \frac{\|x\|}{\sqrt{n}}.$$

The argument of the squareroot in the middle expression is called the *mean-square* value of $x$, denoted $\mathbf{ms}(x)$, and the RMS value is the squareroot of the mean-square value. The RMS value of a vector $x$ is useful when comparing norms of vectors with different dimensions; the RMS value tells us what a 'typical' value of $|x_i|$ is. For example, the norm of $\mathbf{1}$, the $n$-vector of all ones, is $\sqrt{n}$, but its RMS value is 1, independent of $n$. More generally, if all the entries of a vector are the same, say, $\alpha$, then the RMS value of the vector is $|\alpha|$.

**Norm of a sum.** A useful formula for the norm of the sum of two vectors $x$ and $y$ is

$$\|x + y\| = \sqrt{\|x\|^2 + 2x^T y + \|y\|^2}. \qquad (2.1)$$

To derive this formula, we start with the square of the norm of $x+y$ and use various properties of the inner product:

$$\begin{array}{rcl}\|x+y\|^2 & = & (x+y)^T(x+y)\\ & = & x^Tx + x^Ty + y^Tx + y^Ty\\ & = & \|x\|^2 + 2x^Ty + \|y\|^2.\end{array}$$

Taking the squareroot of both sides yields the formula (2.6) above. In the first line, we use the definition of the norm. In the second line, we expand the inner product. In the fourth line we use the definition of the norm, and the fact that $x^Ty = y^Tx$. Several other identities relating norms, sums, and inner products of vectors are explored in the exercises.

**Norm of block vectors.** The norm-squared of a stacked vector is the sum of the norm-squared values of its subvectors. For example, with $a = (b, c, d)$ (where $a$, $b$, and $c$ are vectors), we have

$$\|a\|^2 = a^Ta = b^Tb + c^Tc + d^Td = \|b\|^2 + \|c\|^2 + \|d\|^2.$$

This idea is often used in reverse, to express the sum of the norm-squared values of some vectors as the norm-square value of a block vector formed from them.

We can write the equality above in terms of norms as

$$\|(a, b, c)\| = \sqrt{\|a\|^2 + \|b\|^2 + \|c\|^2} = \|(\|a\|, \|b\|, \|c\|)\|.$$

In words: The norm of a stacked vector is the norm of the vector formed from the norms of the subvectors. (Which is quite a mouthful.) The right-hand side of the equation above should be carefully read. The outer norm symbols enclose a 3-vector, with (scalar) entries $\|a\|$, $\|b\|$, and $\|c\|$.

**Chebyshev inequality.** Suppose that $x$ is an $n$-vector, and that $k$ of its entries satisfy $|x_i| \geq a$, where $a > 0$. Then $k$ of its entries satisfy $x_i^2 \geq a^2$. It follows that

$$\|x\|^2 = x_1^2 + \cdots + x_n^2 \geq ka^2,$$

since $k$ of the numbers in the sum are at least $a^2$, and the other $n-k$ numbers are nonnegative. We can conclude that $k \leq \|x\|^2/a^2$, which is called the *Chebyshev inequality*. When $\|x\|^2/a^2 \geq n$, the inequality tells us nothing, since we always have $k \leq n$. In other cases it limits the number of entries in a vector that can be large. For $a > \|x\|$, the inequality is $k \leq \|x\|^2/a^2 < 1$, so we conclude that $k = 0$ (since $k$ is an integer). In other words, no entry of a vector can be larger in magnitude than the norm of the vector.

The Chebyshev inequality is easier to interpret in terms of the RMS value of a vector. We can write it as

$$\frac{k}{n} \leq \left(\frac{\mathbf{rms}(x)}{a}\right)^2, \qquad (2.2)$$

where $k$ is, as above, the number of entries of $x$ with absolute value at least $a$. The left-hand side is the fraction of entries of the vector that are at least $a$ in absolute

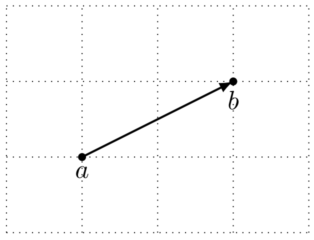

**Figure 2.1** The norm of the displacement $b - a$ is the distance between the points with coordinates $a$ and $b$.

value. The right-hand side is the inverse square of the ratio of $a$ to $\mathbf{rms}(x)$. It says, for example, that no more than $1/25 = 4\%$ of the entries of a vector can exceed its RMS value by more than a factor of 5. The Chebyshev inequality partially justifies the idea that the RMS value of a vector gives an idea of the size of a typical entry: It states that not too many of the entries of a vector can be much bigger than its RMS value.

**Weighted norm.** The *weighted norm* of a vector $x$ is 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') 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. This choice of 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.

## 2.2 Distance

**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 one, two, and three dimensions, this distance is exactly the usual distance between points with coordinates $a$ and $b$, as illustrated in figure 2.1. But the Euclidean distance is defined for vectors of any dimension; we can refer to the distance between two vectors of dimension 100. Since we only use the Euclidean norm in this book, we will refer to the Euclidean distance between vectors as,

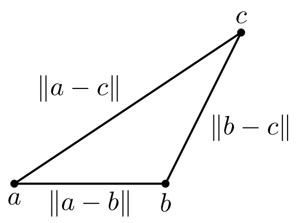

**Figure 2.2** Triangle inequality.

simply, the distance between the vectors. If $a$ and $b$ are $n$-vectors, we refer to the RMS value of the difference, $\|a-b\|/\sqrt{n}$, as the *RMS deviation* between the two vectors.

When the distance between two $n$-vectors $x$ and $y$ is small, we say they are 'close' or 'nearby', and when the distance $\|x-y\|$ is large, we say they are 'far'. The particular numerical values of $\|x-y\|$ that correspond to 'close' or 'far' depend on the particular application.

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

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

Geometric intuition tells us that the length of any side of a triangle cannot exceed the sum of the lengths of the other two sides. For example, we have

$$\|a-c\| \leq \|a-b\| + \|b-c\|. \tag{2.3}$$

This follows from the triangle inequality, since

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

This is illustrated in figure 2.2.

**Examples.**

- *Feature distance.* If $x$ and $y$ represent vectors of $n$ features of two objects, the quantity $\|x-y\|$ is called the *feature distance*, and gives a measure of how different the objects are (in terms of the feature values).

- *Document similarity.* If $n$-vectors $x$ and $y$ represent the histograms of word occurrences for two documents, $\|x-y\|$ represents a measure of the dissimilarity of the two documents. We might expect the dissimilarity to be small when the two documents have the same genre, topic, or author; we would expect it to be larger when they are on different topics, or have different authors.

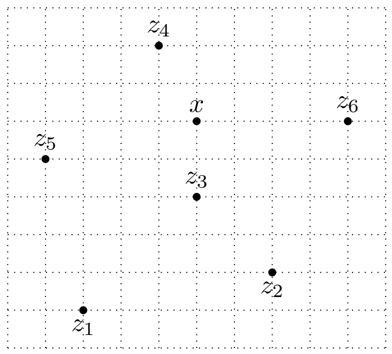

**Figure 2.3** The point $z_3$ is the nearest neighbor of $x$ among the points $z_1$, …, $z_6$.

- *RMS prediction error.* Suppose that the $n$-vector $x$ represents a time series of some quantity, for example, hourly temperature at some location, and $\hat{x}$ is another $n$-vector that represents an estimate or prediction of the time series $x$, based on other information. The difference $\hat{x} - x$ is called the *prediction error*, and its RMS value $\mathbf{rms}(\hat{x} - x)$ is called the *RMS prediction error.* If this value is small (say, compared to $\mathbf{rms}(x)$) the prediction is good.

- *Nearest neighbor.* Suppose $z_1, \ldots, z_k$ is a collection of $k$ $n$-vectors, and that $x$ is another $n$-vector. We say that $z_j$ is the *nearest neighbor* (among all the $z_i$) of $x$ if
$$\|x - z_j\| \leq \|x - z_i\|, \quad i = 1, \ldots, k.$$
In words: $z_j$ is the closest vector to $x$ among the set of vectors. This is illustrated in figure 2.3. The idea of nearest neighbor is used in many applications. For example, suppose the vectors $z_i$ are the word histograms for a collection of $k$ documents, $x$ is the word histogram of a new document, and $z_j$ is the nearest neighbor of $x$. This suggests that the new document is most similar to the document $j$ in our original collection (in terms of histogram distance).

**Standard deviation.** For any vector $x$, the vector $\tilde{x} = x - \mathbf{avg}(x)\mathbf{1}$ is called the associated *de-meaned* vector, obtained by subtracting the mean of the entries from each entry of $x$. The mean value of the entries of $\tilde{x}$ is zero, *i.e.*, $\mathbf{avg}(\tilde{x}) = 0$. This explains why $\tilde{x}$ is called the de-meaned version of $x$; roughly speaking, it is $x$ with its mean removed. The de-meaned vector is useful for understanding how the entries of a vector deviate from their mean value. It is zero if all the entries in the original vector $x$ are the same.

The *standard deviation* of an $n$-vector $x$ is defined as the RMS value of the de-meaned vector $x - \mathbf{avg}(x)\mathbf{1}$, *i.e.*,

$$\mathbf{std}(x) = \sqrt{\frac{(x_1 - \mathbf{avg}(x))^2 + \cdots + (x_n - \mathbf{avg}(x))^2}{n}}.$$

This is the same as the RMS deviation between a vector $x$ and the vector all of whose entries are $\mathbf{avg}(x)$. It can be written using the inner product and norm as

$$\mathbf{std}(x) = \frac{\|x - (\mathbf{1}^T x/n)\mathbf{1}\|}{\sqrt{n}}. \tag{2.4}$$

The standard deviation of a vector $x$ tells us the typical amount by which its entries deviate from their average value. The standard deviation of a vector is zero only when all its entries are equal. The standard deviation of a vector is small when the entries of the vector are nearly the same.

We should warn the reader that another slightly different definition of the standard deviation of a vector is widely used, in which the denominator $\sqrt{n}$ in (2.4) is replaced with $\sqrt{n-1}$. In this book we will only use the definition (2.4).

The average, RMS value, and standard deviation of a vector are related by the formula

$$\mathbf{rms}(x)^2 = \mathbf{avg}(x)^2 + \mathbf{std}(x)^2. \tag{2.5}$$

This formula makes sense: $\mathbf{rms}(x)^2$ is the mean square value of the entries of $x$, which can be expressed as the square of the mean value, plus the mean square fluctuation of the entries of $x$ around their mean value. We can derive this formula from our vector notation formula for $\mathbf{std}(x)$ given above. We have

$$\begin{array}{rcl}
\mathbf{std}(x)^2 & = & (1/n)\|x - (\mathbf{1}^T x/n)\mathbf{1}\|^2 \\
& = & (1/n)(x^T x - 2x^T(\mathbf{1}^T x/n)\mathbf{1} + ((\mathbf{1}^T x/n)\mathbf{1})^T((\mathbf{1}^T x/n))\mathbf{1}) \\
& = & (1/n)(x^T x - (2/n)(\mathbf{1}^T x)^2 + n(\mathbf{1}^T x/n)^2 \\
& = & (1/n)x^T x - (\mathbf{1}^T x/n)^2 \\
& = & \mathbf{rms}(s)^2 - \mathbf{avg}(x)^2,
\end{array}$$

which can be re-arranged to obtain the identity above. This derivation uses many of the properties for norms and inner products, and should be read carefully to understand every step. In the second line, we expand the norm-square of the sum of two vectors. In the third line we use the commutative property of scalar-vector multiplication, moving scalars such as $(\mathbf{1}^T x/n)$ to the front of each term, and also the fact that $\mathbf{1}^T\mathbf{1} = n$.

**Examples.**

- *Mean return and risk.* Suppose that an $n$-vector represents a time series of return on an investment, expressed as a percentage, in $n$ time periods over some interval of time. Its average gives the mean return over the whole interval, often shortened to its *return*. Its standard deviation is a measure of how variable the return is, from period to period, over the time interval, *i.e.*, how much it typically varies from its mean, and is often called the *risk* of the investment over the time interval. Multiple investments can be compared by plotting them on a *risk-return plot*, which gives the mean and standard deviation of the returns of each of the investments over some interval. Figure 2.4 shows an example.

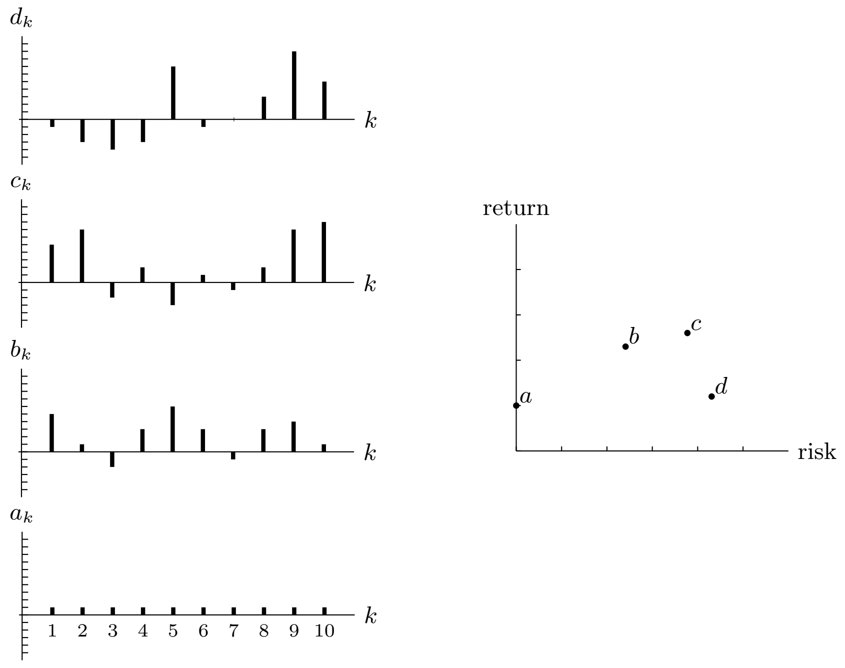

**Figure 2.4** The vectors $a$, $b$, $c$, $d$ represent time series of returns on investments over 10 periods. The right-hand plot shows the investments in a risk-return plane, with return defined as the average value and risk as the standard deviation of the corresponding vector.

- *Temperature or rainfall.* Suppose that the $n$-vector is a time series of the daily average temperature at a particular location, over a one year period. Its average gives the average temperature at that location (over the year) and its standard deviation is a measure of how much the temperature varied from its average values. We would expect the average temperature to be high and the standard deviation to be low in a tropical location, and the opposite for a location with high latitude.

**Chebyshev inequality for standard deviation.** The Chebyshev inequality (2.2) can be transcribed to the standard deviation: If $k$ is the number of entries of $x$ that satisfy $|x_i - \mathbf{avg}(x)| \geq a$, then $k/n \leq (\mathbf{std}(x)/a)^2$. (This inequality is only interesting for $a > \mathbf{std}(x)$.) For example, at most $1/9 = 11.1\%$ of the entries of a vector can deviate from the mean value $\mathbf{avg}(x)$ by 3 standard deviations or more. Another way state this is: The fraction of entries of $x$ within $\alpha$ standard deviations of $\mathbf{avg}(x)$ is at least $1 - 1/\alpha^2$ (for $\alpha > 1$).

As an example, consider a time series of return on an investment, with a mean return of 8%, and a risk (standard deviation) 3%. By the Chebyshev inequality, the fraction of periods with a loss (*i.e.*, $x_i \leq 0$) is no more than $(3/8)^2 = 14.1\%$. (In fact, the fraction of periods when the return is either a loss, $x_i \leq 0$, or very good, $x_i \geq 16\%$, is together no more than 14.1%.)

**Complexity.** Computing the norm of an $n$-vector requires $n$ multiplications (to square each entry), $n-1$ additions (to add the squares), and one squareroot. Even though computing the squareroot typically takes more time than computing the product or sum of two numbers, it is counted as just one flop. So computing the norm takes $2n$ flops. The cost of computing the RMS value is the same, since we can ignore the two flops involved in division by $\sqrt{n}$. Computing the distance between two vectors costs $3n$ flops, and computing the standard deviation costs $4n$ flops. All of these operations have order $n$.

As a slightly more involved computation, suppose that we wish to determine the nearest neighbor among a set of $k$ $n$-vectors $z_1, \ldots, z_k$ to another $n$-vector $x$. The simple approach is to compute the distances $\|x - z_i\|$, for $i = 1, \ldots, k$, and then find the minimum of these. (Sometimes a comparison of two numbers is also counted as a flop.) The cost of this is $3kn$ flops to compute the distances, and $k - 1$ comparisons to find the minimum. The latter term can be ignored, so the flop count is $3kn$. The order of finding the nearest neighbor in a set of $k$ $n$-vectors is $kn$.

**Minimizing the sum of squared distances.** Here we derive a simple result that we will use below. Suppose we are given $N$ $n$-vectors $x_1, x_2, \ldots, x_N$. (Note that here, the subscript $i$ in $x_i$ refers to the $i$th $n$-vector, not the $i$th component of a single $n$-vector $x$.) We wish to select a single $n$-vector $z$ that is near all the given vectors. To do this, we will choose $z$ to minimize the sum of squared distances to the vectors $x_1, \ldots, x_N$, *i.e.*,

$$\|x_1 - z\|^2 + \cdots + \|x_N - z\|^2.$$

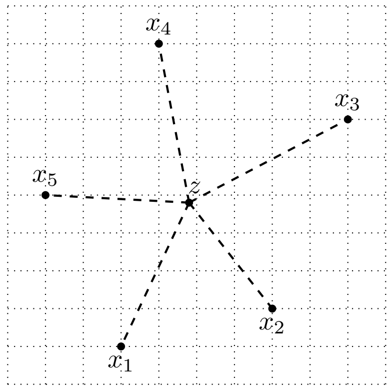

**Figure 2.5** The vector $z$ is the mean or centroid of the vectors $x_1$, …, $x_5$.

(There are other objectives we could use to choose $z$, but we will see that this one has a particularly simple solution.) We can write out the objective in terms of the coefficients of the vectors, as

$$\begin{aligned}\sum_{i=1}^N \|x_i - z\|^2 &= \sum_{i=1}^N \left( \sum_{j=1}^n ((x_i)_j) - z_j)^2 \right) \\ &= \sum_{j=1}^n \left( \sum_{i=1}^N ((x_i)_j) - z_j)^2 \right),\end{aligned}$$

where $(x_i)_j$ refers to the $j$th component of the vector $x_i$. The bottom line above shows that the objective is a sum of terms, each of which involves $z_j$ but none of the other $z_l$. To minimize the sum, we can simply choose the scalar $z_j$ to minimize its term, *i.e.*,

$$\sum_{i=1}^N ((x_i)_j) - z_j)^2.$$

This is done by choosing $z_j$ to be the average or mean of the set of $N$ numbers $(x_i)_1, \ldots, (x_N)_j$:

$$z_j = (1/N)\left((x_1)_1 + \cdots + (x_N)_j\right).$$

(This can be verified directly, as in exercise **??**, or derived from simple calculus.) We can express the solution in compact vector notation as

$$z = (1/N)\left(x_1 + \cdots + x_N\right).$$

We refer to $z$ as the mean or average of the $N$ vectors $x_1, \ldots, x_N$. The mean of a set of vectors is the vector that minimizes the sum of its distance squared to the vectors in the set. Another term for the mean vector $z$ is the *centroid* of the vectors $x_1, \ldots, x_N$. This is illustrated in figure 2.5.

## 2.3 Angle

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

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

for any $n$-vectors $a$ and $b$. Written out in terms of the entries, this is

$$|a_1 b_1 + \cdots + a_n b_n| \leq \left(a_1^2 + \cdots + a_n^2\right)^{1/2} \left(b_1^2 + \cdots + b_n^2\right)^{1/2},$$

which looks more intimidating.

The Cauchy-Schwarz inequality 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). So we suppose now that $a \neq 0$, $b \neq 0$, and define $\alpha = \|a\|$, $\beta = \|b\|$. We observe that

$$\begin{aligned} 0 &\leq \|\beta a - \alpha b\|^2 \\ &= \|\beta a\|^2 - 2(\beta a)^T(\alpha b) + \|\alpha b\|^2 \\ &= \beta^2\|a\|^2 - 2\beta\alpha(a^T b) + \alpha^2\|b\|^2 \\ &= \beta^2\|a\|^2 - 2\beta\alpha(a^T b) + \alpha^2\|b\|^2 \\ &= 2\|a\|^2\|b\|^2 - 2\|a\|\,\|b\|(a^T b). \end{aligned}$$

Dividing by $2\|a\|\,\|b\|$ yields $a^T b \leq \|a\|\,\|b\|$. Applying this inequality to $-a$ and $b$ we obtain $-a^T b \leq \|a\|\,\|b\|$. Putting these two inequalities together we get the Cauchy-Schwarz inequality, $|a^T b| \leq \|a\|\,\|b\|$.

This argument also reveals the conditions on $a$ and $b$ under which they satisfy the Cauchy-Schwarz inequality with equality. This occurs only if $\|\beta a - \alpha b\| = 0$, *i.e.*, $\beta a = \alpha b$. This means that each vector is a scalar multiple of the other. This statement remains true when either $a$ or $b$ is zero. So the Cauchy-Schwarz inequality holds with equality when one of the vectors is a multiple of the other; in all other cases, it holds with strict inequality.

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

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

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

**Angle between vectors.** The *angle* between two nonzero vectors $a$, $b$ is defined as

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

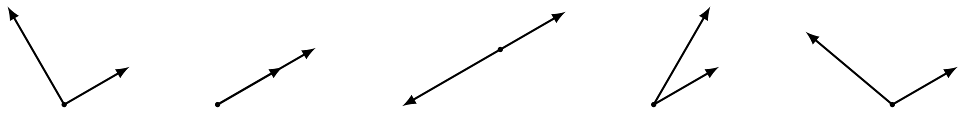

**Figure 2.6** From left to right: examples of orthogonal, aligned, and anti-aligned vectors, vectors that make an acute and an obtuse angle.

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

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

The angle between $a$ and $b$ is written as $\angle(a, b)$, and 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. For example, the angle between the vectors $a = (1, 2, -1)$ and $b = (2, 0, -3)$ is

$$\arccos\left(\frac{5}{\sqrt{6}\,\sqrt{13}}\right) = \arccos(0.5661) = 0.9690 = 55.52^\circ$$

(to within 4 digits). But the definition of angle is more general; we can refer to the angle between two vectors with dimension 100.

The angle is a symmetric function of $a$ and $b$: we have $\angle(a, b) = \angle(b, a)$. The angle is not affected by scaling each of the vectors by a positive scalar: we have, for any vectors $a$ and $b$, and any positive numbers $\alpha$ and $\beta$,

$$\angle(\alpha a, \beta b) = \angle(a, b).$$

**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*. 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*. 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*. 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 $a^T b \geq 0$, *i.e.*, the vectors have nonnegative inner product.
- If $\angle(a, b) \geq \pi/2 = 90^\circ$, the vectors are said to make an *obtuse angle*. This is the same as $a^T b \leq 0$, *i.e.*, the vectors have nonpositive inner product.

These definitions are illustrated in figure 2.6.

**Examples.**

- *Spherical distance.* Suppose $x$ and $y$ are 3-vectors that represent two points that lie on a sphere of radius $R$ (for example, locations on earth). The distance between them, measured along the sphere, is given by $R\angle(x,y)$.
- *Document similarity via angles.* If $n$-vectors $x$ and $y$ represent the word counts for two documents, their angle $\angle(x,y)$ can be used as a measure of document dissimilarity. (When using angle to measure document dissimilarity, either word counts or histograms can be used; they produce the same result.)

**Norm of sum via angles.** For vectors $x$ and $y$ we have

$$\|x+y\|^2 = \|x\|^2 + 2x^Ty + \|y\|^2 = \|x\|^2 + 2\|x\|\|y\|\cos\theta + \|y\|^2, \qquad (2.6)$$

where $\theta = \angle(x,y)$. (The first equality comes from (2.6).) From this we can make several observations.

- If $x$ and $y$ are aligned ($\theta = 0$), we have $\|x+y\| = \|x\| + \|y\|$. Thus, their norms add.
- If $x$ and $y$ are orthogonal ($\theta = 90^\circ$), we have $\|x+y\|^2 = \|x\|^2 + \|y\|^2$. In this case the norm-squared values add, and we have $\|x+y\| = \sqrt{\|x\|^2 + \|y\|^2}$.

**Correlation coefficient.** Suppose $a$ and $b$ are vectors of the same size, with associated de-meaned vectors

$$\tilde{a} = a - \mathbf{avg}(a)\mathbf{1}, \qquad \tilde{b} = b - \mathbf{avg}(b)\mathbf{1}.$$

Assuming these de-meaned vectors are not zero (which occurs when the original vectors have all equal entries), we define their *correlation coefficient* as

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

Thus, $\rho = \cos\theta$, where $\theta = \angle(\tilde{a},\tilde{b})$.

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 $\rho = 0.3$. When $\rho = 0$, we say the vectors are *uncorrelated.*

The correlation coefficient tells us how the entries in the two vectors vary together. Roughly speaking, high correlation (say, $\rho = 0.8$) means that entries of $a$ and $b$ are typically above their mean for many of the same entries. The extreme case $\rho = 1$ occurs only if the vectors $\tilde{a}$ and $\tilde{b}$ are aligned, which means that each is a positive multiple of the other, and the other extreme case $\rho = -1$ occurs only when $\tilde{a}$ and $\tilde{b}$ are negative multiples of each other. This idea is illustrated in figure 2.7, which shows the entries of two vectors, as well as a scatter plot of them, for cases with correlation near 1, near $-1$, and near 0.

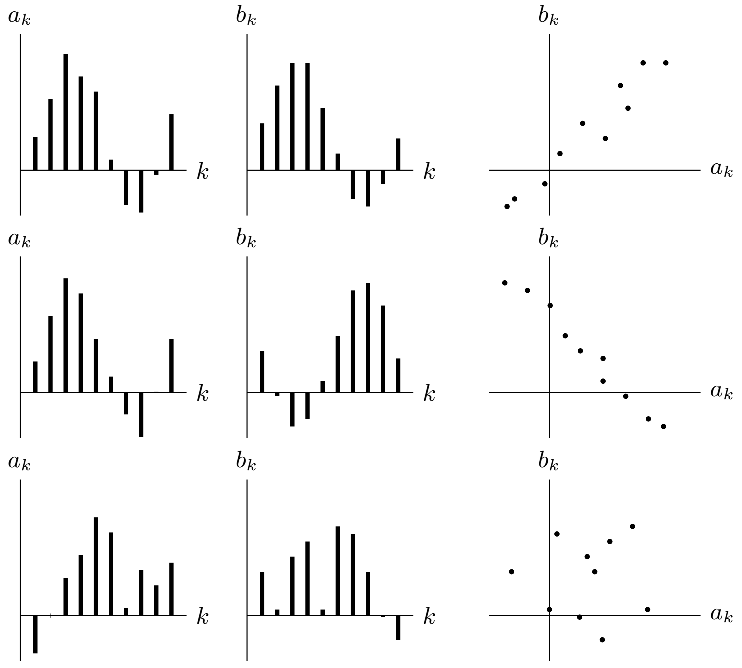

**Figure 2.7** Three pairs of vectors $a$, $b$ of length 10, with correlation coefficients 0.968 (top), −0.988 (middle), and 0.004 (bottom).

The correlation coefficient is often used when the vectors represent time series, such as the returns on two investments over some time interval, or the rainfall in two locations over some time interval. If they are highly correlated (say, $\rho > 0.8$), the two time series are typically above their mean values at the same times. For example, we would expect the rainfall time series at two nearby locations to be highly correlated. As another example, we might expect the returns of two similar companies, in the same business area, to be highly correlated.

## 2.4 The $k$-means algorithm

In this section we use the ideas introduced above to decribe a widely used method for clustering or partitioning a set of $N$ $n$-vectors into $k$ groups, with the vectors in each group near other. Normally we have $k \ll N$, which means that there are many more vectors than groups. Typical applications use values of $k$ that range from a handful to a few hundred or more, with values of $N$ that range from hundreds to billions.

Clustering is very widely used in many application areas, typically (but not always) when the vectors represent features of objects. We start with the $n$-vectors $x_1, x_2, \ldots, x_N$. The goal is to divide or partition these $N$ vectors into $k$ groups (also called clusters) of vectors, where vectors in each group are close to each other.

**Partitioning the vectors given the representatives.** To carry out the partitioning, we introduce a *representative* $n$-vector for each of the $k$ groups, which we denote $z_1, \ldots, z_k$. These representatives can be any $n$-vectors; they do not need to be one of the given vectors. We will say later how we find these representatives.

Given the representatives, we can partition or cluster the original vectors by assigning each $x_i$ to group $j$, if $z_j$ is the nearest representative to $x_i$. (If there is a tie for the nearest representative, we can assign $x_i$ to any of the groups associated with a closest representative.) So, given the representatives, we can carry out the clustering of the original set of vectors in a natural way. This is illustrated in the left-hand half of figure 2.8.

**Choosing the representatives given the partitioning.** Now we consider the converse problem: Given the clustering of the original vectors into $k$ groups, how should we choose the representative for each group? A very reasonable choice is to choose the representative for cluster $j$ as the vector that minimizes the sum of the squared distances to the vectors in the cluster. We saw above (page 29) that the vector that minimizes the sum of the squared distances to a set of vectors is simply the mean or centroid of the vectors. So, given the clustering, we can choose the representatives as the mean or centroid of the vectors in its cluster. For this reason the representatives are also called the *cluster centroids*. This is illustrated in the right-hand half of figure 2.8.

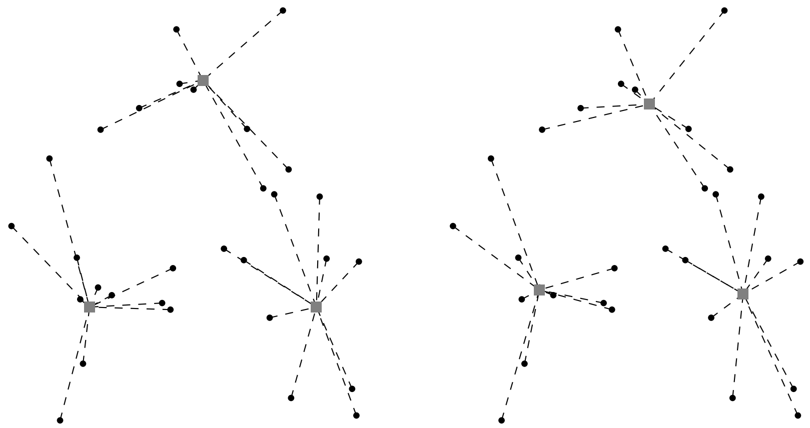

**Figure 2.8** *One iteration of the k-means algorithm.* The 30 points shown as filled circles are partitioned into three groups, with representatives shown as rectangles. In the left-hand figure the representatives are given and each point is assigned to the group with the nearest representative. On the right-hand side, the clusters are the same as on the left, but the representatives are replaced by the cluster centroids.

**The $k$-means algorithm.** The two methods above are circular: One tells us how to partition or cluster the points, given the representatives, and the other tells us how to choose the representatives, given the partition. The *k-means algorithm* chooses an initial set of representatives or centroids, and then simply alternates between these two operations until it converges, which occurs when successive partitions are the same.

**Algorithm 2.1** $k$-MEANS ALGORITHM

```
given a set of $N$ vectors $x_1, \dots, x_N$, and an initial set of $k$ centroid vectors $z_1, \dots, z_k$

repeat until convergence

    1. Partition the vectors into $k$ groups. For each vector $i = 1, \dots, N$,
       assign $x_i$ to the group associated with the nearest representative.
    2. Update representatives. For each group $j = 1, \dots, k$,
       set $z_j$ to be the mean of the vectors in group $j$.
```

(Ties in step 1 can be broken by assigning $x_i$ to the group associated with one of the closest representatives with the smallest value of $j$.)

**Quality of partition.** One way to judge the quality of a partition is to evaluate the quantity

$$J = (1/N) \sum_{i=1}^{N} \min_{j=1,\dots,k} \|x_i - z_j\|^2,$$

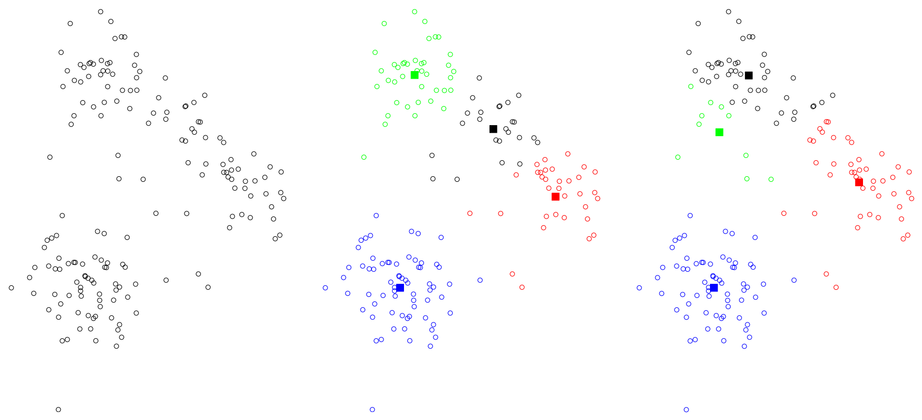

**Figure 2.9** A set of 200 points and their partitioning into 4 clusters by the $k$-means algorithm started at two different starting points.

which is the mean squared distance from the original vectors to their closest representatives. If this number is small, it means that the vectors have been partitioned into $k$ groups, where the vectors in each group are close to their associated representative, and therefore, each other. Each step in the $k$-means algorithm reduces (or at least, does not increase) $J$.

**Convergence.** The fact that $J$ decreases in each step can be used to show that the $k$-means algorithm converges: After a finite number of steps, the partition in two subsequent steps is the same. No changes in the means or the partition will occur after that, even if the algorithm is continued. (See exercise **??**.)

However, depending on the initial choice of representatives, the algorithm can, and does, converge to different final partitions, with different quality measures. The $k$-means algorithm is a *heuristic*, which means it cannot guarantee that the partition it finds minimizes $J$. For this reason it is common to run the $k$-means algorithm several times, with different initial representatives, and choose the one among them with the smallest final value of $J$. Figure 2.9 shows an initial set of vectors, and the resulting partitions found using the $k$-means algorithm, started with two different initial sets of centroids. Figure 2.10 shows the value of $J$ versus iteration for the two cases.

Despite the fact that the $k$-means algorithm is a heuristic, it is very useful in practical applications, and very widely used.

**Examples.**

- *Topic discovery.* Suppose $x_i$ are word histograms associated with $N$ documents. The $k$-means algorithm partitions the documents into $k$ groups, which typically can be interpreted as groups of documents with the same or similar topics, genre, or author. Since the $k$-means algorithm runs automatically and without any understanding of what the words in the dictionary mean, this is sometimes called *automatic topic discovery*.

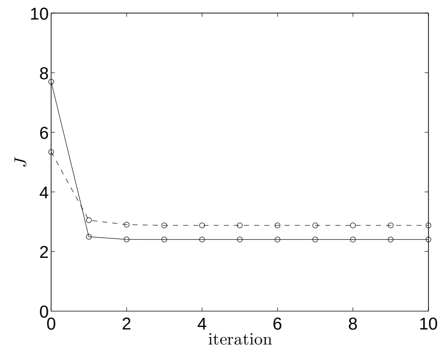

**Figure 2.10** Cost function $J$ versus iteration number for the two starting points in figure 2.9.

- *Patient clustering.* If $x_i$ are feature vectors associated with $N$ patients admitted to a hospital, the $k$-means algorithm clusters the patients into $k$ groups of similar patients (at least in terms of their feature vectors).
- *Customer market segmentation.* Suppose the vector $x_i$ gives the quantities of $n$ items purchased by customer $i$ over some period of time. The $k$-means algorithm will group the customers into $k$ market segments, which are groups of customers with similar purchasing patterns.
- *ZIP-code clustering.* Suppose that $x_i$ is a vector giving $n$ quantities or statistics for the residents of ZIP-code $i$, such as numbers of residents in various age groups, household size, education statistics, and income statistics. (In this example $N$ is around 40000.) The $k$-means algorithm might be used to cluster the 40000 ZIP-codes into, say, $k = 100$ groups of ZIP-codes with similar statistics.
- *Student clustering.* Suppose the vector $x_i$ gives the detailed grading record of student $i$ in a course, *i.e.*, her grades on each question in the quizzes, homework assignments, and exams. The $k$-means algorithm might be used to cluster the students into $k = 10$ groups of students who performed similarly.
- *Survey response clustering.* A group of $N$ people respond to a survey with $n$ questions. Each question contains a statement, such as 'The movie was too long', followed by a set of ordered options such as

  Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree.

  (This is called a *Likert scale.*) Suppose the $n$-vector $x_i$ encodes the selections of respondent $i$ on the $n$ questions, using the numerical coding $-2$, $-1$, 0, $+1$, $+2$ for the responses above. The $k$-means algorithm can be used to cluster the respondents into $k$ groups, each with similar responses to the survey. Note that the final representative vectors found can have entries that are not integers. For example, we might have $(z_2)_3 = 2.6$, which means that people in cluster 2 quite strongly agreed with the statement in question 3.

**Choosing $k$.** It is common to run the $k$-means algorithm for different values of $k$, and compare the results. How to choose a value of $k$ among these depends on how the clustering will be used. But some general statements can be made. For example, if the value of $J$ with $k = 7$ is quite a bit smaller than for the values $k = 2, \ldots, 6$, and not much larger than for the values $k = 8, 9, \ldots$, we could reasonably choose $k = 7$, and conclude that our data (set of vectors) partitions nicely into 7 groups.

**Understanding data via clustering.** Clustering can give insight or lead to an understanding of a data set (of vectors). For example, it might be useful to know that voters in some election can be well clustered into 7 groups, on the basis of a data set that includes demographic data and questionaire or poll data. The associated representative vectors are quite interpretable: If the 4th component of our vectors is the age of the voter, then $(z_3)_4 = 37.8$ tells that the average of voters in group 3 is 37.8. Insight gained from this data can be used to tune campaign messages, or choose media outlets for campaign advertising.

**Prediction using clustering.** Once we have partitioned a data set consisting of $N$ vectors into $k$ groups, we can use the representatives to assign additional (or new) vectors into the existing groups. For example, suppose we run $k$-means on a large enough set of documents to get a clustering into topics that appears to be useful. Given a new document, not in our original corpus of documents, we can assign it to the topic group with the closest representative.

**Complexity of $k$-means algorithm.** In step 1 of the $k$-means algorithm, we find the nearest neighbor to each of $N$ $n$-vectors, over the set of $k$ centroids. This requires approximately $3Nkn$ flops. In step 2 we average the $n$-vectors over each of cluster groups. For a cluster with $p$ vectors, this requires $n(p-1)$ flops, which we approximate as $np$ flops; averaging all clusters requires a total of $Nn$ flops. This is negligible compared to the cost of patitioning in step 1. So $k$-means requires around $3Nkn$ flops per iteration. Its order is $Nkn$ flops.

Each run of $k$-means typically takes fewer than a few tens of iterations, and usually $k$-means is run some modest number of times, like 10. So a very rough guess of the number of flops required to run $k$-means 10 times (in order to choose the best partition found) is $1000Nkn$ flops. As an example, suppose we use $k$-means to partition $N = 100000$ vectors with size $n = 100$ into $k = 10$ groups. On a 1 Gflop computer we guess that this will take around 100 seconds, *i.e.*, on the order of one minute. Given the approximations made here (for example, the number of iterations that each run of $k$-means will take) this is obviously a crude estimate.

# Chapter 3

# Linear independence

In this chapter we explore the concept of linear independence, which will play an important role in the sequel.

## 3.1 Linear dependence

A set of $n$-vectors $\{a_1, \ldots, a_k\}$ (with $k \geq 1$) is called *linearly dependent* if

$$\beta_1 a_1 + \cdots + \beta_k a_k = 0$$

holds for some $\beta_1, \ldots, \beta_k$, that are not all zero. In other words, we can form the zero vector as a linear combination of the vectors, with coefficients that are not all zero.

When a set of vectors is linearly dependent, at least one of the vectors can be expressed as a linear combination of the other vectors: if $\beta_i \neq 0$ in the equation above (and by definition, this must be true for at least one $i$), we can move the term $\beta_i a_i$ to the other side of the equation and divide by $\beta_i$ to get

$$a_i = (-\beta_1/\beta_i)a_1 + \cdots + (-\beta_{i-1}/\beta_i)a_{i-1} + (-\beta_{i+1}/\beta_i)a_{i+1} + \cdots + (-\beta_k/\beta_i)a_k.$$

The converse is also true: If any vector in a set of vectors is a linear combination of the other vectors, then the set of vectors is linearly dependent.

Following standard mathematical language usage, we will say "The vectors $a_1, \ldots, a_k$ are linearly dependent" to mean "$\{a_1, \ldots, a_k\}$ is a linearly dependent set of vectors". But it must be remembered that linear dependence is an attribute of a *set* of vectors, and not individual vectors.

**Linearly independent set of vectors.** A set of $n$-vectors $\{a_1, \ldots, a_k\}$ (with $k \geq 1$) is called *linearly independent* if it is not linearly dependent, which means that

$$\beta_1 a_1 + \cdots + \beta_k a_k = 0 \qquad (3.1)$$

only holds for $\beta_1 = \cdots = \beta_k = 0$. In other words, the only linear combination of the vectors that equals the zero vector is the linear combination with all coefficients zero.

As with linear dependence, we will say "The vectors $a_1, \ldots, a_k$ are linearly independent" to mean "$\{a_1, \ldots, a_k\}$ is a linearly independent set of vectors". But, like linear dependence, linear independence is an attribute of a set of vectors, and not individual vectors.

It is generally not easy to determine by casual inspection whether or not a set of vectors is linearly dependent or linearly independent. But we will soon see an algorithm that does this.

**Examples.**

- Any set of vectors containing the zero vector is linearly dependent.
- A set of two vectors is linearly dependent if and only if one of the vectors is a multiple of the other one.
- The vectors

$$a_1 = \begin{bmatrix} 0.2 \\ -7 \\ 8.6 \end{bmatrix}, \qquad a_2 = \begin{bmatrix} -0.1 \\ 2 \\ -1 \end{bmatrix}, \qquad a_3 = \begin{bmatrix} 0 \\ -1 \\ 2.2 \end{bmatrix},$$

  are linearly dependent, since $a_1 + 2a_2 - 3a_3 = 0$. We can express any of these vectors as a linear combination of the other two. For example, we have $a_2 = (-1/2)a_1 + (3/2)a_3$.
- The standard unit $n$-vectors $e_1, \ldots, e_n$ are linearly independent. To see this, suppose that (3.1) holds. We have

$$0 = \beta_1 e_1 + \cdots + \beta_n e_n = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix},$$

  so we conclude that $\beta_1 = \cdots = \beta_n = 0$.

**Linear combinations of linearly independent vectors.** Suppose a vector $x$ is a linear combination of $a_1, \ldots, a_k$,

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

When the vectors $a_1, \ldots, a_k$ are linearly independent, the coefficients that form $x$ are *unique*: If we also have

$$x = \gamma_1 a_1 + \cdots + \gamma_k a_k,$$

then $\beta_i = \gamma_i$, $i = 1, \ldots, k$. This tells us that, in principle at least, we can find the coefficients that form a vector $x$ as a linear combination of linearly independent vectors.

To see this, we subtract the two equations above to get

$$0 = (\beta_1 - \gamma_1)a_1 + \cdots + (\beta_1 - \gamma_k)a_k.$$

Since $a_1, \ldots, a_k$ are linearly independent, we conclude that $\beta_i - \gamma_i$ are all zero.

The converse is also true: If each linear combination of a set of vectors can only be expressed as a linear combination with one set of coefficients, then the set of vectors is independent. This gives a nice interpretation of linear independence: A set of vectors in linearly independent if and only if for any linear combination of them, we can infer or deduce the associated coefficients.

**Supersets and subsets.** If a set of vectors is linearly dependent, then any superset of it is linearly dependent. (In other words: If we add vectors to a linearly dependent set of vectors, the new set is linearly dependent.) Any nonempty subset of a linearly independent set of vectors is linearly independent. (In other words: Removing vectors from a set of vectors preserves linear independence.)

## 3.2 Basis

**Independence-dimension inequality.** If the $n$-vectors $a_1, \ldots, a_k$ are linearly independent, then $k \leq n$. In words:

*A linearly independent set of $n$-vectors can have at most $n$ elements.*

Put another way:

*Any set of $n+1$ or more $n$-vectors is linearly dependent.*

We will prove this fundamental fact below; but first, we describe the concept of basis, which relies on the independence-dimension inequality.

**Basis.** A set of $n$ linearly independent $n$-vectors (*i.e.*, a set of linearly independent vectors of the maximum possible size) is called a *basis*. If the $n$-vectors $a_1, \ldots, a_n$ are a basis, then any $n$-vector $b$ can be written as a linear combination of them. To see this, consider the set of $n+1$ $n$-vectors $\{a_1, \ldots, a_n, b\}$. By the independence-dimension inequality, these vectors are linearly dependent, so there are $\beta_1, \ldots, \beta_{n+1}$, not all zero, that satisfy

$$\beta_1 a_1 + \cdots + \beta_n a_n + \beta_{n+1} b = 0.$$

If $\beta_{n+1} = 0$, then we have

$$\beta_1 a_1 + \cdots + \beta_n a_n = 0,$$

which, since $a_1, \ldots, a_n$ are linearly independent, implies that $\beta_1 = \cdots = \beta_n = 0$. But then all the $\beta_i$ are zero, a contradiction. So we conclude that $\beta_{n+1} \neq 0$. It follows that

$$b = (-\beta_1/\beta_{n+1})a_1 + \cdots + (-\beta_n/\beta_{n+1})a_n,$$

*i.e.*, $b$ is a linear combination of $a_1, \ldots, a_n$.

Combining this result with the observation above that any linear combination of linearly independent vectors can be expressed in only one way, we conclude:

*Any $n$-vector $b$ can be written in a unique way as a linear combination of a basis $a_1, \ldots, a_n$.*

**Proof of independence-dimension inequality.** The proof is by induction on $n$.

First consider a linearly independent set $\{a_1, \ldots, a_k\}$ of 1-vectors. We must have $a_1 \neq 0$. This means that every element $a_i$ of the set can be expressed as a multiple $a_i = (a_i/a_1)a_1$ of the first element $a_1$. This contradicts linear independence unless $k = 1$.

Next suppose $n \geq 2$ and the independence-dimension inequality holds for dimension $n-1$. Let $\{a_1, \ldots, a_k\}$ be a linearly independent set of $n$-vectors. We need to show that $k \leq n$. We partition the vectors as

$$a_i = \left[\begin{array}{c} b_i \\ \alpha_i \end{array}\right], \quad i = 1, \ldots, k,$$

where $b_i$ is an $(n-1)$-vector and $\alpha_i$ is a scalar.

First suppose that $\alpha_1 = \cdots = \alpha_k = 0$. Then the set $\{b_1, \ldots, b_k\}$ is linearly independent: $\sum_{i=1}^k \beta_i b_i = 0$ holds if and only if $\sum_{i=1}^k \beta_i a_i = 0$, which is only possible for $\beta_1 = \cdots = \beta_k = 0$ because the vectors $a_i$ are linearly independent. The vectors $b_1, \ldots, b_k$ therefore form a linearly independent set of $(n-1)$-vectors. By the induction hypothesis we have $k \leq n-1$, so certainly $k \leq n$.

Next suppose that the scalars $\alpha_i$ are not all zero. Assume $\alpha_j \neq 0$. We define a set of $k-1$ vectors $c_i$ of length $n-1$ as follows:

$$c_i = b_i - \frac{\alpha_i}{\alpha_j} b_j, \quad i = 1, \ldots, j-1, \qquad c_i = b_{i+1} - \frac{\alpha_{i+1}}{\alpha_j} b_j, \quad i = j, \ldots, k-1.$$

These $k-1$ vectors are linearly independent: if $\sum_{i=1}^{k-1} \beta_i c_i = 0$ then

$$\sum_{i=1}^{j-1} \beta_i \left[\begin{array}{c} b_i \\ \alpha_i \end{array}\right] + \gamma \left[\begin{array}{c} b_j \\ \alpha_j \end{array}\right] + \sum_{i=j+1}^{k} \beta_{i-1} \left[\begin{array}{c} b_i \\ \alpha_i \end{array}\right] = 0 \qquad (3.2)$$

with

$$\gamma = -\frac{1}{\alpha_j}\left(\sum_{i=1}^{j-1} \beta_i \alpha_i + \sum_{i=j+1}^{k} \beta_{i-1} \alpha_i\right).$$

Since the vectors $a_i = (b_i, \alpha_i)$ are linearly independent, the equality (3.2) only holds when all the coefficients $\beta_i$ and $\gamma$ are all zero. This in turns implies that the vectors $c_1, \ldots, c_{k-1}$ are linearly independent. By the induction hypothesis $k - 1 \leq n - 1$, so we have established that $k \leq n$.

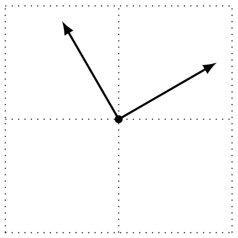

**Figure 3.1** Orthonormal vectors in a plane.

## 3.3 Orthonormal set of vectors

A set of vectors $\{a_1, \ldots, a_k\}$ is *orthogonal* or *mutually orthogonal* if $a_i \perp a_j$ for any $i$, $j$ with $i \neq j$, $i, j = 1, \ldots, k$. A set of vectors $\{a_1, \ldots, a_k\}$ is *orthonormal* if it is orthogonal and $\|a_i\| = 1$ for $i = 1, \ldots, k$. (A vector of norm one is called *normalized*; dividing a vector by its norm is called *normalizing* it.) Thus, each vector in an orthonormal set of vectors is normalized, and any pair of two different vectors from the set are orthogonal. These two conditions can be combined into one statement about the inner products of pairs of vectors in the set: $\{a_1, \ldots, a_k\}$ is orthonormal means that

$$a_i^T a_j = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

Orthonormality, like linear dependence and independence, is an attribute of a set of vectors, and not an attribute of vectors individually. By convention, though, we say "$a_1, \ldots, a_k$ are orthonormal" to mean "$\{a_1, \ldots, a_k\}$ is an orthonormal set".

**Examples.** The standard unit $n$-vectors $e_1, \ldots, e_n$ are orthonormal. As another example, the 3-vectors

$$\left[\begin{array}{c} 0 \\ 0 \\ -1 \end{array}\right], \qquad \frac{1}{\sqrt{2}}\left[\begin{array}{c} 1 \\ 1 \\ 0 \end{array}\right], \qquad \frac{1}{\sqrt{2}}\left[\begin{array}{c} 1 \\ -1 \\ 0 \end{array}\right],$$

are orthonormal. Figure 3.1 shows a set of two orthonormal 2-vectors.

**Independence of orthonormal vectors.** Orthonormal vectors are linearly independent. To see this, suppose $a_1, \ldots, a_k$ are orthonormal, and

$$\beta_1 a_1 + \cdots + \beta_k a_k = 0.$$

Taking the inner product of this equality with $a_i$ yields

$$\begin{aligned} 0 &= a_i^T(\beta_1 a_1 + \cdots + \beta_k a_k) \\ &= \beta_1(a_i^T a_1) + \cdots + \beta_k(a_i^T a_k) \\ &= \beta_i, \end{aligned}$$

since $a_i^T a_j = 0$ for $j \neq i$ and $a_i^T a_i = 1$. Thus, the only linear combination of $a_1, \ldots, a_k$ that is zero is the one with all coefficients zero.

**Linear combinations of orthonormal vectors.** Suppose a vector $x$ is a linear combination of $a_1, \ldots, a_k$, where $a_1, \ldots, a_k$ are orthonormal,

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

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

$$a_i^T x = a_i^T(\beta_1 a_1 + \cdots + \beta_k a_k) = \beta_i,$$

using the same argument as above. So if a vector $x$ is a linear combination of orthonormal vectors, we can easily find the coefficients of the linear combination by taking the inner products with the vectors.

For any $x$ that is a linear combination of orthonormal vectors $a_1, \ldots, a_k$, we have the identity

$$x = (a_1^T x)a_1 + \cdots + (a_k^T x)a_k. \tag{3.3}$$

This identity gives us a simple way to check if an $n$-vector $y$ is a linear combination of the orthonormal vectors $a_1, \ldots, a_k$. If the identity (3.3) holds for $y$, *i.e.*,

$$y = (a_1^T y)a_1 + \cdots + (a_k^T y)a_k,$$

then (evidently) $y$ is a linear combination of $a_1, \ldots, a_k$; conversely, if $y$ is a linear combination of $a_1, \ldots, a_k$, the identity (3.3) holds.

**Orthonormal basis.** If the $n$-vectors $a_1, \ldots, a_n$ are orthonormal, they are independent, and therefore also a basis. In this case they are called an *orthonormal basis*. The three examples above (on page 45) are orthonormal bases.

If $a_1, \ldots, a_n$ are an orthonormal basis, then we have, for any $n$-vector $x$, the identity

$$x = (a_1^T x)a_1 + \cdots + (a_n^T x)a_n. \tag{3.4}$$

To see this, we note that since $a_1, \ldots, a_n$ are a basis, $x$ can be expressed as a linear combination of them; hence the identity (3.3) above holds. The equation above is sometimes called the *orthonormal expansion formula*; the right-hand side is called the *expansion of $x$ in the basis* $a_1, \ldots, a_n$. It shows that any $n$-vector can be expressed as a linear combination of the basis elements, with the coefficients given by taking the inner product of $x$ with the elements of the basis.

## 3.4 Gram-Schmidt algorithm

The Gram-Schmidt algorithm can be used to determine if a set of $n$-vectors $\{a_1, \ldots, a_k\}$ is linearly independent. In later chapters, we will see that it has many other uses as well.

If the vectors are linearly independent, it produces an orthonormal set of vectors $\{q_1, \ldots, q_k\}$ with the following properties: for each $i = 1, \ldots, k$, $a_i$ is a linear combination of $q_1, \ldots, q_i$, and $q_i$ is a linear combination of $a_1, \ldots, a_i$. If the vectors $\{a_1, \ldots, a_j\}$ are linearly independent, but $\{a_1, \ldots, a_{j+1}\}$ are linearly dependent, the algorithm detects this and terminates. In other words, it finds the first vector $a_j$ that is a linear combination of previous vectors $a_1, \ldots, a_{j-1}$.

**Algorithm 3.1** GRAM-SCHMIDT ALGORITHM

```
given n-vectors $a_1, \ldots, a_k$
for $i = 1, \ldots, k$,
    1. Orthogonalization. $\tilde{q}_i = a_i - (q_1^T a_i)q_1 - \cdots - (q_{i-1}^T a_i)q_{i-1}$
    2. Test for dependence. if $\tilde{q}_i = 0$, quit.
    3. Normalization. $q_i = \tilde{q}_i / \|\tilde{q}_i\|$
```

The orthogonalization step, with $i = 1$, reduces to $\tilde{q}_1 = a_1$. If the algorithm does not quit (in step 2), *i.e.*, $\tilde{q}_1, \ldots, \tilde{q}_k$ are all nonzero, we can conclude that the original set of vectors is independent; if the algorithm does quit early, say, with $\tilde{q}_j = 0$, we can conclude that the original set of vectors is dependent (and indeed, that $a_j$ is a linear combination of $a_1, \ldots, a_{j-1}$).

**Analysis of Gram-Schmidt algorithm.** Let us show that the following hold, for $i = 1, \ldots, k$, assuming $a_1, \ldots, a_k$ are linearly independent.

1. $\tilde{q}_i \neq 0$, so the dependence test in step 2 is not satisfied, and we do not have a divide-by-zero error in step 3.
2. $q_1, \ldots, q_i$ are orthonormal.
3. $a_i$ is a linear combination of $q_1, \ldots, q_i$.
4. $q_i$ is a linear combination of $a_1, \ldots, a_i$.

We show this by induction. For $i = 1$, we have $\tilde{q}_1 = a_1$. Since $a_1, \ldots, a_k$ are independent, we must have $a_1 \neq 0$, and therefore $\tilde{q}_1 \neq 0$, so assertion 1 holds. The set $\{q_1\}$ is evidently orthonormal, since $\|q_1\| = 1$, so assertion 2 holds. We have $a_1 = \|\tilde{q}_1\| q_1$, and $q_1 = (1/\|\tilde{q}_1\|)a_1$, so assertions 3 and 4 hold.

Suppose our assertion holds for some $i - 1$, with $i < k$; we will show it holds for $i$. If $\tilde{q}_i = 0$, then $a_i$ is a linear combination of $q_1, \ldots, q_{i-1}$ (from the first step in the algorithm); but each of these is (by the induction hypothesis) a linear combination of $a_1, \ldots, a_{i-1}$, so it follows that $a_i$ is a linear combination of $a_1, \ldots, a_{i-1}$, which contradicts our assumption that $a_1, \ldots, a_k$ are independent. So assertion 1 holds for $i$.

Step 3 of the algorithm ensures that $q_1, \ldots, q_i$ are normalized; to show they are orthogonal we will show that $q_i \perp q_j$ for $j = 1, \ldots, i-1$. (Our induction hypothesis tells us that $q_r \perp q_s$ for $r, s < i$.) For any $j = 1, \ldots, i-1$, we have (using step 1 of the algorithm)

$$\begin{aligned} q_j^T \tilde{q}_i &= q_j^T a_i - (q_1^T a_i)(q_j^T q_1) - \cdots - (q_{i-1}^T a_i)(q_j^T q_{i-1}) \\ &= q_j^T a_i - q_j^T a_i = 0, \end{aligned}$$

using $q_j^T q_k = 0$ for $j \neq k$ and $q_j^T q_j = 1$. (This explains why step 1 is called the orthogonalization step: We subtract from $a_i$ a linear combination of $q_1, \ldots, q_{i-1}$ that ensures $q_i \perp \tilde{q}_j$ for $j < i$.) Since $q_i = (1/\|\tilde{q}_i\|)q_i$, we have $q_i^T q_j = 0$ for $j = 1, \ldots, i-1$. So assertion 2 holds for $i$.

It is immediate that $a_i$ is a linear combination of $q_1, \ldots, q_i$:

$$\begin{array}{rcl} a_i & = & \tilde{q}_i + (q_1^T a_i)q_1 + \cdots + (q_{i-1}^T a_i)q_{i-1} \\ & = & (q_1^T a_i)q_1 + \cdots + (q_{i-1}^T a_i)q_{i-1} + \|\tilde{q}_i\| q_i. \end{array}$$

From step 1 of the algorithm, we see that $\tilde{q}_i$ is a linear combination of $a_1, q_1, \ldots, q_{i-1}$. By the induction hypothesis, each of $q_1, \ldots, q_{i-1}$ is a linear combination of $a_1, \ldots, a_{i-1}$, so $\tilde{q}_i$ (and therefore also $q_i$) is a linear combination of $a_1, \ldots, a_i$. Thus assertions 3 and 4 hold.

**Gram-Schmidt completion implies independence.** From the properties 1–4 above, we can argue that the original set of vectors $a_1, \ldots, a_k$ is linearly independent. To see this, suppose that

$$\beta_1 a_1 + \cdots + \beta_k a_k = 0 \tag{3.5}$$

holds for some $\beta_1, \ldots, \beta_k$. We will show that $\beta_1 = \cdots = \beta_k = 0$.

We first note that any linear combination of $q_1, \ldots, q_{k-1}$ is orthogonal to any multiple of $q_k$, since $q_1^T q_k = \cdots = q_{k-1}^T q_k = 0$ (by definition). But each of $a_1, \ldots, a_{k-1}$ is a linear combination of $q_1, \ldots, q_{k-1}$, so we have $q_k^T a_1 = \cdots = q_k^T a_{k-1} = 0$. Taking the inner product of $q_k$ with the left- and right-hand sides of (3.5) we obtain

$$\begin{array}{rcl} 0 & = & q_k^T(\beta_1 a_1 + \cdots + \beta_k a_k) \\ & = & \beta_1 q_k^T a_1 + \cdots + \beta_{k-1} q_k^T a_{k-1} + \beta_k q_k^T a_k \\ & = & \beta_k \|\tilde{q}_k\|, \end{array}$$

where we use $q_k^T a_k = \|\tilde{q}_k\|$ in the last line. We conclude that $\beta_k = 0$.

From (3.5) and $\beta_k = 0$ we have

$$\beta_1 a_1 + \cdots + \beta_{k-1} a_{k-1} = 0.$$

We now repeat the argument above to conclude that $\beta_{k-1} = 0$. Repeating it $k$ times we conclude that all $\beta_i$ are zero.

**Early termination.** Suppose that the Gram-Schmidt algorithm terminates prematurely, in iteration $j$, because $\tilde{q}_j = 0$. The conclusions 1–4 above hold for $i = 1, \ldots, j-1$, since in those steps $\tilde{q}_i$ is nonzero. Since $\tilde{q}_j = 0$, we have

$$a_j = (q_1^T a_j)q_1 - \cdots - (q_{j-1}^T a_j)q_{j-1},$$

which shows that $a_j$ is a linear combination of $q_1, \ldots, q_{j-1}$. But each of these vectors is in turn a linear combination of $a_1, \ldots, a_{j-1}$, by conclusion 3 above. Then $a_j$ is a linear combination of $a_1, \ldots, a_{j-1}$, since it is a linear combination of linear combinations of them (see page 10). This means that $a_1, \ldots, a_j$ are linearly dependent, which implies that the larger set $a_1, \ldots, a_k$ are linearly dependent.

In summary, the Gram-Schmidt algorithm gives us an explicit method for determining if a set of vectors is dependent or independent.

**Example.** We define three vectors

$$a_1 = (-1,1,-1,1), \qquad a_2 = (-1,3,-1,3), \qquad a_3 = (1,3,5,7).$$

Applying the Gram-Schmidt algorithm gives the following results.

- $i=1$. We have $\|\tilde{q}_1\| = 2$, so

  $$q_1 = \frac{1}{\|\tilde{q}_1\|}\tilde{q}_1 = (-1/2, 1/2, -1/2, 1/2),$$

  which is simply $a_1$ normalized.

- $i=2$. We have $q_1^T a_2 = 4$, so

  $$\tilde{q}_2 = a_2 - (q_1^T a_2)q_1 = \begin{bmatrix} -1 \\ 3 \\ -1 \\ 3 \end{bmatrix} - 4\begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

  which is indeed orthogonal to $q_1$ (and $a_1$). It has norm $\|\tilde{q}_2\| = 2$; normalizing it gives

  $$q_2 = \frac{1}{\|\tilde{q}_2\|}\tilde{q}_2 = (1/2, 1/2, 1/2, 1/2).$$

- $i=3$. We have $q_1^T a_3 = 2$ and $q_2^T a_3 = 8$, so

  $$\begin{aligned} \tilde{q}_3 &= a_3 - (q_1^T a_3)q_1 - (q_2^T a_3)q_2 \\ &= \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} - 2\begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} - 8\begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} \\ &= \begin{bmatrix} -2 \\ -2 \\ 2 \\ 2 \end{bmatrix}, \end{aligned}$$

  which is orthogonal to $q_1$ and $q_2$ (and $a_1$ and $a_2$). We have $\|\tilde{q}_3\| = 4$, so the normalized vector is

  $$q_3 = \frac{1}{\|\tilde{q}_3\|}\tilde{q}_3 = (-1/2, -1/2, 1/2, 1/2).$$

Completion of the Gram-Schmidt algorithm without early termination tells us that the vectors $a_1, a_2, a_3$ are linearly independent.

**Determining if a vector is a linear combination of independent vectors.** Suppose the vectors $a_1, \ldots, a_k$ are linearly independent, and we wish to determine if another vector $b$ is a linear combination of them. (We have already noted that if it is a linear combination of them, the coefficients are unique.) The Gram-Schmidt algorithm provides an explicit way to do this. We apply the Gram-Schmidt algorithm to the list of vectors

$$a_1, \ldots, a_k,\ b.$$

This set of vectors is dependent if $b$ is a linear combination of $a_1, \ldots, a_k$; it is independent if $b$ is not a linear combination of $a_1, \ldots, a_k$. The Gram-Schmidt algorithm will determine which of these two cases holds. It cannot terminate in the first $k$ steps, since we assume that $a_1, \ldots, a_k$ are linearly independent. It will terminate in the $(k+1)$st step with $\tilde{q}_{k+1} = 0$ if $b$ is a linear combination of $a_1, \ldots, a_k$. It will not terminate in the $(k+1)$st step (*i.e.*, $\tilde{q}_{k+1} \neq 0$), otherwise.

**Checking if a collection of vectors is a basis.** To check if a set of $n$ $n$-vectors $a_1, \ldots, a_n$ is a basis, we run the Gram-Schmidt algorithm on them. If Gram-Schmidt terminates early, they are not a basis; if it runs to completion, we know they are a basis.

**Complexity of the Gram-Schmidt algorithm.** We now derive an operation count for the Gram-Schmidt algorithm. In the first step of iteration $i$ of the algorithm, $i-1$ inner products

$$q_1^T a_i, \ldots, q_{i-1}^T a_i$$

between vectors of length $n$ are computed. This takes $(i-1)(2n-1)$ flops. We then use these inner products as the coefficients in $i-1$ scalar multiplications with the vectors $q_1, \ldots, q_{i-1}$. This requires $n(i-1)$ flops. We then subtract the $i-1$ resulting vectors from $a_i$, which requires another $n(i-1)$ flops. The total flop count for step 1 is

$$(i-1)(2n-1) + n(i-1) + n(i-1) = (4n-1)(i-1)$$

flops. In step 3 we compute the norm of $\tilde{q}_i$, which takes approximately $2n$ flops. We then divide $\tilde{q}_i$ by its norm, which requires $n$ scalar divisions. So the total flop count for the $i$th iteration is $(4n-1)(i-1) + 3n$ flops.

The total flop count for all $k$ iterations of the algorithm is obtained by summing our counts for $i = 1, \ldots, k$:

$$\sum_{i=1}^{k} ((4n-1)(i-1) + 3n) = (4n-1)\frac{k(k-1)}{2} + 3nk \approx 2nk^2,$$

where we use the fact that $\sum_{i=1}^{k}(i-1) = k(k-1)/2$. The complexity of the Gram-Schmidt algorithm is $2nk^2$; its order is $nk^2$. We can guess that its running time grows linearly with the lengths of the vectors $n$, and quadratically with the number of vectors $k$.

In the special case of $k = n$, the complexity of the Gram-Schmidt method is $2n^3$. For example, if the Gram-Schmidt algorithm is used to determine whether a set of $n = 1000$ 1000-vectors is independent (and therefore a basis), the computational cost is around $2 \times 10^9$ flops. On a modern computer, can we can expect this to take on the order of one second.

When the Gram-Schmidt algorithm is implemented, a variation on it called the *modified Gram-Schmidt* algorithm is typically used. This algorithm produces the same results as the Gram-Schmidt algorithm (3.1), but is less sensitive to the small round-off errors that occur when arithmetic calculations are done using floating-point numbers. (We do not consider round-off error in this book.)

# Chapter 4

# Matrices

In this chapter we introduce matrices and some basic operations on them. We give some applications in which they arise, and explain their connection to solving systems of linear equations.

## 4.1 Matrices

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

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

It is also common to use large parentheses instead of rectangular brackets, as in

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

An important attribute of a matrix is its *size* or *dimensions*, *i.e.*, the numbers of rows and columns. The matrix above has 3 rows and 4 columns, so its size is $3 \times 4$. 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 $i$th row and $j$th column, denoted by double subscripts: the $i, j$ element of a matrix $A$ is denoted $A_{ij}$ (or $A_{i,j}$, when $i$ or $j$ is more than one digit or character). The positive integers $i$ and $j$ are called the (row and column) *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 $n$.

If the matrix above is $B$, then we have $B_{13} = -2.3$, $B_{32} = -1$. The row index of the bottom left element (which has value 4.1) is 3; its column index is 1.

Two matrices are equal if they have the same size, and the corresponding entries are all equal. As with vectors, we normally deal with matrices with entries that are real numbers, which will be our assumption unless we state otherwise. The set of real $m \times n$ matrices is denoted $\mathbf{R}^{m \times n}$. But matrices with complex entries, for example, do arise in some applications.

**Matrix indexing.** As with vectors, standard mathematical notation indexes the rows and columns of a matrix starting from 1. In computer languages, matrices are often (but not always) stored as 2-dimensional arrays, which can be indexed in a variety of ways, depending on the language. Lower level languages typically use indexes starting from 0; higher level languages and packages that support matrix operations usually use standard mathematical indexing, starting from 1.

**Square, tall, and wide matrices.** 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 $n > m$).

**Column and row vectors.** 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 called a *row vector*; to give its size, we can refer to it as an $n$-row-vector. As an example,

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

is a row vector (or $1 \times 3$ matrix). To distinguish them from row vectors, vectors are sometimes called *column vectors*. A $1 \times 1$ matrix is considered to be the same as a scalar.

**Columns and rows of a matrix.** An $m \times n$ matrix $A$ has $n$ columns, given by (the $m$-vectors)

$$a_j = \begin{bmatrix} A_{1j} \\ \vdots \\ A_{mj} \end{bmatrix},$$

for $j = 1, \ldots, n$. The same matrix has $m$ rows, given by the ($1 \times n$ row vectors)

$$b_i = \begin{bmatrix} A_{i1} & \cdots & A_{in} \end{bmatrix},$$

for $i = 1, \ldots, m$.

**Block matrices and submatrices.** It is useful to consider matrices whose entries are themselves matrices, as in

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

where $B$, $C$, $D$, and $E$ are matrices. Such matrices are called *block matrices*; the elements $B$, $C$, and $D$ are called *blocks* or *submatrices* of $A$. The submatrices can be referred to by their block row and column indices; for example, $C$ is the 1,2 block of $A$.

Block matrices must have the right dimensions 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 example above, $B$ and $C$ must have the same number of rows, and $C$ and $E$ must have the same number of columns.

As an example, consider

$$B = \left[\begin{array}{ccc} 0 & 2 & 3 \end{array}\right], \qquad C = \left[\begin{array}{c} -1 \end{array}\right], \qquad D = \left[\begin{array}{ccc} 2 & 2 & 1 \\ 1 & 3 & 5 \end{array}\right], \qquad E = \left[\begin{array}{c} 4 \\ 4 \end{array}\right].$$

Then the block matrix $A$ above is given by

$$A = \left[\begin{array}{cccc} 0 & 3 & 3 & -1 \\ 2 & 2 & 1 & 4 \\ 1 & 3 & 5 & 4 \end{array}\right].$$

(Note that we have dropped the left and right brackets that delimit the blocks. This is similar to the way we drop the brackets in a $1 \times 1$ matrix to get a scalar.)

Matrix blocks placed next to each other in the same row are said to be *concatenated*; matrix blocks placed above each other are called *stacked*.

We can also divide a larger matrix (or vector) into 'blocks'. In this context the blocks are called *submatrices* of the big matrix. As with vectors, we can use colon notation to denote submatrices. If $A$ is an $m \times n$-matrix, and $p$, $q$, $r$, $s$ are integers with $1 \leq p \leq q \leq m$ and $1 \leq r \leq s \leq n$, then $A_{p:q,r:s}$ denotes the submatrix

$$A_{p:q,r:s} = \left[\begin{array}{cccc} A_{pr} & A_{p,r+1} & \cdots & A_{ps} \\ A_{p+1,r} & A_{p+1,r+1} & \cdots & A_{p+1,s} \\ \vdots & \vdots & & \vdots \\ A_{qr} & A_{q,r+1} & \cdots & A_{qs} \end{array}\right].$$

This submatrix has size $(q-p+1) \times (s-r+1)$ and is obtained by extracting from $A$ the elements in rows $p$ through $q$ and columns $r$ through $s$.

**Column and row representation of a matrix.** Using block matrix notation we can write an $m \times n$ matrix $A$ as a block matrix with one block row and $n$ block columns,

$$A = \left[\begin{array}{cccc} a_1 & a_2 & \cdots & a_n \end{array}\right],$$

where $a_j$, which is a $m$-vector, is the $j$th column of $A$. Thus, an $m \times n$ matrix can be viewed as its $n$ columns, concatenated.

Similarly, an $m \times n$ matrix $A$ can be written as a block matrix with one block column and $m$ block rows:

$$A = \left[\begin{array}{c} b_1 \\ b_2 \\ \vdots \\ b_m \end{array}\right],$$

where $b_i$, which is a row $n$-vector, is the $i$th row of $A$. In this notation, the matrix $A$ is interpreted as its $m$ rows, stacked.

## 4.2 Examples

**Table interpretation.** The most direct interpretation of a matrix is as a table of numbers that depend on two indexes, $i$ and $j$. (A vector is a list of numbers that depend on only one index.) In this case the rows and columns of the matrix usually have some simple interpretation. Some examples are given below.

- *Images.* A black and white image with $M \times N$ pixels is naturally represented as an $M \times N$ matrix. The row index $i$ gives the vertical position of the pixel, the column index $j$ gives the horizontal position of the pixel, and the $i, j$ entry gives the pixel value.

- *Rainfall data.* An $m \times n$ matrix $A$ gives the rainfall at $m$ different locations on $n$ consecutive days, so $A_{42}$ (which is a number) is the rainfall at location 4 on day 2. The $j$th column of $A$, which is an $m$-vector, gives the rainfall at the $m$ locations on day $j$. The $i$th row of $A$, which is an $n$-row-vector, is the time series of rainfall at location $i$.

- *Asset returns.* A $T \times n$ matrix $R$ gives the returns of a collection of $n$ assets (called the *universe* of assets) over $T$ periods, with $R_{ij}$ giving the return of asset $j$ in period $i$. $R_{12,7} = -0.03$ means that asset 7 had a 3% loss in period 12. The 4th column of $R$ is a $T$-vector that is the return time series for asset 4. The 3rd row of $R$ is an $n$-row-vector that gives the returns of all assets in the universe in period 3.

- *Prices from multiple suppliers.* An $m \times n$ matrix $P$ gives the prices of $n$ different goods from $m$ different suppliers (or locations): $P_{ij}$ is the price that supplier $i$ charges for good $j$. The $j$th column $P$ is the $m$-vector of supplier prices for good $j$; the $i$th row gives the prices for all goods from supplier $i$.

- *Contingency table.* Suppose we have a collection of objects with two attributes, the first attribute with $m$ possible values and the second with $n$ possible values. An $m \times n$ matrix $A$ can be used to hold the counts of the numbers of objects with the different pairs of attributes: $A_{ij}$ is the number of objects with first attribute $i$ and second attribute $j$. (This is the analog of a count $n$-vector, that records the counts of one attribute in a collection.) For example, a population of college students can be described by a $4 \times 50$ matrix, with the $i, j$ entry the number of students in year $i$ of their studies, from state $j$ (with the states ordered in, say, alphabetical order). The $i$th row of $A$ gives the geographic distribution of students in year $i$ of their studies; the $j$th column of $A$ is a 4-vector giving the numbers of student from state $j$ in their first through fourth years of study.

**Matrix representation of a collection of vectors.** Matrices are very often used as a compact way to give a set of indexed vectors of the same size. For example, if $x_1, \ldots, x_N$ are $n$-vectors that give the $n$ feature values for each of $N$ objects, we can collect them all into one $n \times N$ matrix

$$X = \left[ \begin{array}{cccc} x_1 & x_2 & \cdots & x_N \end{array} \right],$$

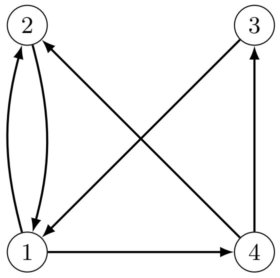

**Figure 4.1** The relation (4.1) as a directed graph.

often called a *data matrix* or *feature matrix*. Its $j$th column is the feature $n$-vector for the $j$th object (in this context sometimes called the $j$th *example*). The $i$th row of the data matrix $X$ is an $m$-row-vector whose entries are the values of the $j$th feature across the examples. We can also directly interpret the entries of the data matrix: $X_{ij}$ (which is a number) is the value of the $i$th feature for the $j$th example.

As another example, a $3 \times M$ matrix can be used to represent a collection of $M$ locations or positions in 3 dimensional space, with its $j$th column giving the $j$th position.

**Matrix representation of a relation or graph.** Suppose we have $n$ objects labeled $1, \ldots, n$. A *relation* $\mathcal{R}$ on the set of objects $\{1, \ldots, n\}$ is a subset of ordered pairs of objects. As an example, $\mathcal{R}$ can represent a *preference relation* among $n$ possible products or choices, with $(i, j) \in \mathcal{R}$ meaning that choice $i$ is preferred to choice $j$. A relation can also be viewed as a *directed graph*, with nodes (or vertices) labeled $1, \ldots, n$, and a directed edge from $j$ to $i$ for each $(i, j) \in \mathcal{R}$. This is typically drawn as a graph, with arrows indicating the direction of the edge, as shown in figure 4.1, for the relation on 4 objects

$$\mathcal{R} = \{(1,2),\ (1,3),\ (2,1),\ (2,4),\ (3,4),\ (4,1)\}. \qquad (4.1)$$

A relation $\mathcal{R}$ on $\{1, \ldots, n\}$ is represented by the $n \times n$ matrix $A$ with

$$A_{ij} = \left\{ \begin{array}{ll} 1 & (i,j) \in \mathcal{R} \\ 0 & (i,j) \notin \mathcal{R}. \end{array} \right.$$

The relation (4.1), for example, is represented by the matrix

$$A = \left[ \begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{array} \right].$$

## 4.3 Zero and identity matrices

**Zero matrix.** 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.

**Identity matrix.** 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

$$I_{ij} = \left\{ \begin{array}{ll} 1 & i = j \\ 0 & i \neq j. \end{array} \right.$$

For example,

$$\left[ \begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right], \qquad \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]$$

are the $2 \times 2$ and $4 \times 4$ 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

$$I = \left[ \begin{array}{cccc} e_1 & e_2 & \cdots & e_n \end{array} \right],$$

where $e_k$ is the $k$th unit vector of size $n$.

Sometimes a subscript is used to denote the size of an 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

$$A = \left[ \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array} \right],$$

then

$$\left[ \begin{array}{cc} I & A \\ 0 & I \end{array} \right] = \left[ \begin{array}{ccccc} 1 & 0 & 1 & 2 & 3 \\ 0 & 1 & 4 & 5 & 6 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right].$$

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

The importance of the identity matrix will become clear later, in §6.1.

**Sparse matrices.** A matrix $A$ is said to be *sparse* if many of its entries are zero, or (put another way) just a few of its entries are nonzero. Its *sparsity pattern* is the set of indices $(i,j)$ for which $A_{ij} \neq 0$. The *number of nonzeros* of a sparse matrix $A$ is the number of entries in its sparsity pattern, and denoted $\mathbf{nnz}(A)$. If $A$ is $m \times n$ we have $\mathbf{nnz}(A) \leq mn$. Its *density* is $\mathbf{nnz}(A)/(mn)$, which is no more than one. Densities of sparse matrices that arise in applications are typically small or very small, as in $10^{-2}$ or $10^{-4}$. There is no precise definition of how small the density must be for a matrix to qualify as sparse. A famous definition of sparse matrix due to Wilkinson is "A matrix is sparse if it has enough zero entries that it pays to take advantage of them".

Like sparse vectors, sparse matrices arise in many applications, and can be stored and manipulated efficiently on a computer. An $n \times n$ identity matrix is sparse, since it has only $n$ nonzeros, so its density is $1/n$. The zero matrix is the sparsest possible matrix, since it has no nonzeros entries. Several special sparsity patterns have names; we describe some important ones below.

**Diagonal matrices.** A square $n \times n$ matrix $A$ is *diagonal* if $A_{ij} = 0$ for $i \neq j$. (The entries of a matrix with $i = j$ are called the *diagonal entries*; those with $i \neq j$ are its *off-diagonal* entries.) A diagonal matrix is one for which all off-diagonal entries are zero. Examples of diagonal matrices we have already seen are square zero matrices and identity matrices. Other examples are

$$\left[\begin{array}{rr} -3 & 0 \\ 0 & 0 \end{array}\right], \qquad \left[\begin{array}{rrr} 0.2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 1.2 \end{array}\right].$$

(Note that in the first example, one of the diagonal elements is also zero.)

The notation $\mathbf{diag}(a_1, \ldots, a_n)$ is used to compactly describe the $n \times n$ diagonal matrix $A$ with diagonal entries $A_{11} = a_1, \ldots, A_{nn} = a_n$. This notation is not yet standard, but is coming into more prevalent use. As examples, the matrices above would be expressed as

$$\mathbf{diag}(-3, 0), \qquad \mathbf{diag}(0.2, -3, 1.2),$$

respectively. We also allow **diag** to take one $n$-vector argument, as in $I = \mathbf{diag}(\mathbf{1})$.

**Triangular matrices.** A square $n \times n$ matrix $A$ is *upper triangular* if $A_{ij} = 0$ for $i > j$, and it is *lower triangular* if $A_{ij} = 0$ for $i < j$. (So a diagonal matrix is one that is both lower and upper triangular.) If a matrix is either lower or upper triangular, it is called *triangular*.

For example, the matrices

$$\left[\begin{array}{rrr} 1 & -1 & 0.7 \\ 0 & 1.2 & -1.1 \\ 0 & 0 & 3.2 \end{array}\right], \qquad \left[\begin{array}{rr} -0.6 & 0 \\ -0.3 & 3.5 \end{array}\right],$$

are upper and lower triangular, respectively.

A triangular $n \times n$ matrix $A$ has $n(n+1)/2$ nonzero entries, *i.e.*, around half its entries are zero. Triangular matrices are generally not considered sparse matrices, since their density is around 50%, but their special sparsity pattern will be important in the sequel.

## 4.4 Transpose and addition

**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,

$$\left[\begin{array}{cc} 0 & 4 \\ 7 & 0 \\ 3 & 1 \end{array}\right]^T = \left[\begin{array}{ccc} 0 & 7 & 3 \\ 4 & 0 & 1 \end{array}\right].$$

If we transpose a matrix twice, we get back the original matrix: $(A^T)^T = A$. A square matrix $A$ is *symmetric* if $A = A^T$, *i.e.*, $A_{ij} = A_{ji}$ for all $i, j$. (The superscript $T$ in the transpose is the same one used to denote the inner product of two $n$-vectors; we will soon see how they are related.)

Transposition converts row vectors into column vectors and vice versa. It is sometimes convenient to express a row vector as $a^T$, where $a$ is a column vector. For example, we might refer to the $m$ rows of an $m \times n$ matrix $A$ as $\tilde{a}_i^T, \ldots, \tilde{a}_m^T$, where $\tilde{a}_1, \ldots, \tilde{a}_m$ are (column) $n$-vectors. We say that a set of row vectors is linearly dependent (or independent) if their transposes (which are column vectors) are dependent (or independent). For example, 'the rows of a matrix $A$ are linearly indendent' means that the columns of $A^T$ are linearly independent.

**Transpose of block matrix.** The transpose of a block matrix has the simple form (shown here for a $2 \times 2$ block matrix)

$$\left[\begin{array}{cc} A & B \\ C & D \end{array}\right]^T = \left[\begin{array}{cc} A^T & C^T \\ B^T & D^T \end{array}\right],$$

where $A$, $B$, $C$, and $D$ are matrices with compatible sizes. Roughly speaking, the transpose of a block matrix is the transposed block matrix, with each element transposed.

**Document-term matrix.** Consider a corpus (collection) of $N$ documents, with word count vectors for a dictionary with $n$ words. The *document-term* matrix associated with the corpus is the $N \times n$ matrix $A$, with $A_{ij}$ the number of times word $j$ appears in document $i$. The rows of the document-term matrix are $a_1^T, \ldots, a_N^T$, where the $n$-vectors $a_1, \ldots, a_N$ are the word count vectors for documents $1, \ldots, N$, respectively. The columns of the document-term matrix are also interesting. The $j$th column of $A$, which is an $N$-vector, gives the number of times word $j$ appears in the corpus of $N$ documents.

**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,

$$\left[\begin{array}{cc} 0 & 4 \\ 7 & 0 \\ 3 & 1 \end{array}\right] + \left[\begin{array}{cc} 1 & 2 \\ 2 & 3 \\ 0 & 4 \end{array}\right] = \left[\begin{array}{cc} 1 & 6 \\ 9 & 3 \\ 3 & 5 \end{array}\right].$$

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 we can only add or subtract matrices of the same size, $I$ refers to a $2 \times 2$ identity matrix.)

The following important properties of matrix addition can be verified directly from the definition. We assume here that $A$, $B$, and $C$ are matrices of the same size.

- *Commutativity.* $A + B = B + A$.
- *Associativity.* $(A + B) + C = A + (B + C)$. We therefore write both as $A + B + C$.
- *Addition with zero matrix.* $A + 0 = 0 + A = A$. Adding the zero matrix to a matrix has no effect.
- *Transpose of sum.* $(A + B)^T = A^T + B^T$. The transpose of a sum of two matrices is the sum of their transposes.

**Scalar-matrix multiplication.** Scalar multiplication of matrices is defined in a similar way as for vectors, and is done by multiplying every element of the matrix by the scalar. For example

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

As with scalar-vector multiplication, the scalar can also appear on the right. Note that $0\,A = 0$ (where the left-hand zero is the scalar zero, and the right-hand 0 is the zero matrix).

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

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

**Complexity of matrix addition.** The addition of two $m \times n$ matrices or a scalar multiplication of an $m \times n$ matrix take $mn$ flops. When $A$ is sparse, scalar multiplication requires $\mathbf{nnz}(A)$ flops. When at least one of $A$ and $B$ is sparse, computing $A + B$ requires $\min\{\mathbf{nnz}(A), \mathbf{nnz}(B)\}$ flops. (For any entry $i, j$ for which one of $A_{ij}$ or $B_{ij}$ is zero, no arithmetic operations are needed to find $(A + B)_{ij}$.)

## 4.5 Matrix-vector multiplication

If $A$ is an $m \times n$ matrix and $x$ is an $n$-vector, then the *matrix-vector product* $y = Ax$ is the $m$-vector $y$ with elements

$$y_i = \sum_{k=1}^{n} A_{ik} x_k = A_{i1} x_1 + \cdots + A_{in} x_n, \quad i = 1, \ldots, m. \tag{4.2}$$

We can express the matrix-vector product in a number of different ways. From (4.2) we see that $y_i$ is the inner product of $x$ with the $i$th row of $A$:

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

where $b_i^T$ is the row $i$ of $A$. The matrix-vector product can also be interpreted in terms of the columns of $A$. If $a_k$ is the $k$th column of $A$, then $y = Ax$ can be written

$$y = x_1 a_1 + x_2 a_2 + \cdots + x_n a_n.$$

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

**Examples.** In the examples below, $A$ is an $m \times n$ matrix and $x$ is an $n$-vector.

- *Zero matrix.* When $A = 0$, we have $Ax = 0$. In other words, $0x = 0$. (The left-hand 0 is an $m \times n$ matrix, and the right-hand zero is an $m$-vector.)

- *Identity.* We have $Ix = x$ for any vector $x$. (The identity matrix here has dimension $n \times n$.) In other words, multplying a vector by the identity matrix gives the same vector.

- *Picking out columns and rows.* An important identity is $Ae_j = a_j$, the $j$th column of $A$. Multiplying a unit vector by a matrix 'picks out' one of the columns of the matrix. $A^T e_i$, which is an $n$-vector, is the $i$th row of $A$, transposed. (In other words, $(A^T e_i)^T$ is the $i$th row of $A$.)

- *Summing or averaging columns or rows.* The $m$-vector $A\mathbf{1}$ is the sum of the columns of $A$. $A^T\mathbf{1}$ is a column vector that gives the row sums of $A$. The $m$-vector $A(\mathbf{1}/n)$ is the average of the columns of $A$.

**Inner product.** When $a$ and $b$ are $n$-vectors, $a^T b$ is exactly the inner product of $a$ and $b$, obtained from the rules for transposing matrices and matrix-vector product. We start with the $n$-(column) vector $a$, consider it as an $n \times 1$ matrix, and transpose it to obtain the $n$-row-vector $a^T$. Now we multiply this $1 \times n$ matrix by the $n$-vector $b$, which we consider an $n \times 1$ matrix, to obtain the $1 \times 1$ matrix $a^T b$, which we also consider a scalar.

**Linear dependence of columns.** We can express the concepts of linear dependence and independence in a compact form using matrix-vector multiplication. The columns of a matrix $A$ are linearly dependent if $Ax = 0$ for some $x \neq 0$. The columns of a matrix $A$ are linearly independent if $Ax = 0$ implies $x = 0$.

If the columns of $A$ are a basis, which means $A$ is square with linearly independent columns, then for any $n$-vector $b$ we can find an $n$-vector $x$ (in fact, a unique one) that satisfies $Ax = b$.

**Examples.**

- *Feature matrix and weight vector.* Suppose $X$ is a feature matrix, where its $N$ columns $x_1, \ldots, x_N$ are the $n$-feature vectors for $N$ objects or examples. Let the $n$-vector $w$ be a *weight vector*, and let $s_i = x_i^T w$ be the score associated with object $i$ using the weight vector $w$. Then we can write $s = X^T w$, where $s$ is the $N$-vector of scores on the objects.

- *Portfolio return time series.* Suppose that $R$ is a $T \times n$ asset return matrix, that gives the returns of $n$ assets over $T$ periods. Let $h$ denote the $n$-vector of investments in the assets over the whole period, so, *e.g.*, $h_3 = 200$ means that we have invested \$200 in asset 3. (Short positions are denoted by negative entries in $h$.) Then $Rh$, which is a $T$-vector, is the time series of the portfolio profit (in \$) over the periods $1, \ldots, T$. If $w$ is a set of portfolio weights (with $\mathbf{1}^T w = 1$), then $Rw$ is the time series of portfolio returns, given as a fraction.

- *Polynomial evaluation at multiple points.* Suppose the entries of the $n$-vector $c$ are the coefficients of a polynomial $p$ of degree $n-1$ or less:

  $$p(t) = c_n t^{n-1} + c_{n-1} t^{n-2} + \cdots + c_2 t + c_1.$$

  Let $t_1, \ldots, t_m$ be $m$ numbers, and define the $m$-vector $y$ as $y_i = p(t_i)$. Then we have $y = Ac$, where $A$ is the $m \times n$ matrix

  $$A = \left[\begin{array}{ccccc} 1 & t_1 & \cdots & t_1^{n-2} & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-2} & t_2^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & t_m & \cdots & t_m^{n-2} & t_m^{n-1} \end{array}\right]. \qquad (4.3)$$

  So multiplying a vector $c$ by the matrix $A$ is the same as evaluating a polynomial with coefficients $c$ at $m$ points. The matrix $A$ in (4.3) comes up often, and is called a *Vandermonde matrix* (of degree $n-1$, at the points $t_1, \ldots, t_m$).

- *Total price from multiple suppliers.* Suppose the $m \times n$ matrix $P$ gives the prices of $n$ goods from $m$ suppliers (or in $m$ different locations). If $q$ is an $n$-vector of quantities of the $n$ goods (sometimes called a *basket* of goods), then $c = Pq$ is an $N$-vector that gives the total cost of the goods, from each of the $N$ suppliers.

**Input-output interpretation.** We can interpret the relation $y = Ax$, with $A$ an $m \times n$ matrix, as a mapping from the $n$-vector $x$ to the $m$-vector $y$. In this context we might think of $x$ as an input, and $y$ as the corresponding output. From equation (4.2), we can interpret $A_{ij}$ as the factor by which $y_i$ depends on $x_j$. Some examples of conclusions we can draw are given below.

- If $A_{23}$ is positive and large, then $y_2$ depends strongly on $x_3$, and increases as $x_3$ increases.
- If $A_{32}$ is much larger than the other entries in the third row of $A$, then $y_3$ depends much more on $x_2$ than the other inputs.
- If $A$ is square and lower triangular, then $y_i$ only depends on $x_1, \ldots, x_i$.

**Complexity of matrix-vector multiplication.** A matrix-vector multiplication of an $m \times n$ matrix $A$ with an $n$-vector $x$ requires $m(2n-1)$ flops, which we simplify to $2mn$ flops. This can be seen as follows. The result $y = Ax$ of the product is an $m$-vector, so there are $m$ numbers to compute. The $i$th element of $y$ is the inner product of the $i$th row of $A$ and the vector $x$, which takes $2n - 1$ flops.

If the matrix in the product has a simple structure, and is stored in a format that allows us to take advantage of the structure, the matrix-vector multiplication can often be computed more efficiently. For example, multiplying a $n$-vector $x$ with a diagonal matrix $A$ requires only $n$ flops, since $y_i = A_{ii}x_i$ for $i = 1, \ldots, n$.

If $A$ is sparse, computing $Ax$ requires $\mathbf{nnz}(A)$ multiplies (of $A_{ij}$ and $x_j$, for each nonzero entry) and a number of additions that is no more than $\mathbf{nnz}(A)$. Thus, the complexity is around $2\,\mathbf{nnz}(A)$ flops.

## 4.6 Examples

### 4.6.1 Geometric transformations

Suppose that the 3-vector $x$ represents a position in 3-dimensional space. Several important geometric transformations or mappings from points to points can be expressed as matrix-vector products $y = Ax$, with $A$ a $3 \times 3$ matrix. In the examples below, we consider the mapping from $x$ to $y$, and focus on the 2-dimensional case (for which some of the matrices are simpler to describe).

**Scaling.** Scaling is the mapping $y = ax$, where $a$ is a scalar. This can be expressed as $y = Ax$ with $A = aI$. This mapping stretches a vector by the factor $|a|$ (or shrinks it when $|a| < 1$), and it flips the vector (reverses its direction) if $a < 0$.

**Dilation.** Dilation is the mapping $y = Dx$, where $D$ is a diagonal matrix, $D = \mathbf{diag}(d_1, d_2)$. This mapping stretches the vector $x$ by different factors along the two different axes. (Or shrinks, if $|d_i| < 1$, and flips, if $d_i < 0$.)

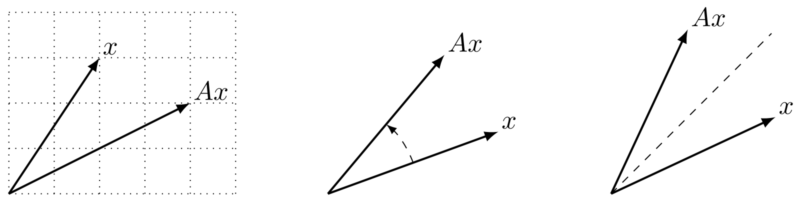

**Figure 4.2** From left to right: a dilation with $A = \mathbf{diag}(2, 2/3)$, a counterclockwise rotation by $\pi/6$ radians, and a reflection through a line that makes an angle of $\pi/4$ radians with the horizontal line.

**Rotation.** Suppose that $y$ is the vector obtained by rotating $x$ by $\theta$ radians counterclockwise. Then we have

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

This matrix is called (for obvious reasons) a *rotation matrix.*

**Reflection.** Suppose that $y$ is the vector obtained by reflecting $x$ through the line that passes through the origin, inclined $\theta$ radians with respect to horizontal. Then we have

$$y = \begin{bmatrix} \cos(2\theta) & \sin(2\theta) \\ \sin(2\theta) & -\cos(2\theta) \end{bmatrix} x.$$

Some of these geometric transformations are illustrated in figure 4.2.

### 4.6.2 Selectors

An $m \times n$ *selector matrix* $A$ is one in which each row is a unit vector (transposed):

$$A = \begin{bmatrix} e_{k_1}^T \\ \vdots \\ e_{k_m}^T \end{bmatrix},$$

where $k_1, \ldots, k_m$ are integers in the range $1, \ldots, n$. When it multiplies a vector, it simply copies the $k_i$th entry of $x$ into the $i$th entry of $y = Ax$:

$$y = (x_{k_1}, x_{k_2}, \ldots, x_{k_m}).$$

The identity matrix, and the reverser matrix are special cases of selector matrices. Another one is the $r\!:\!s$ slicing matrix $A$, which can be described as the block matrix

$$A = \begin{bmatrix} 0_{m\times(r-1)} & I_{m\times m} & 0_{m\times(n-s)} \end{bmatrix},$$

where $m = s - r + 1$. (We show the dimensions of the blocks for clarity.) We have $Ax = x_{r:s}$, *i.e.*, multiplying by $A$ gives the $r\!:\!s$ slice of a vector.

**Down-sampling.** Another example is the $n/2 \times n$ matrix (with $n$ even)

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

If $y = Ax$, we have $y = (x_1, x_3, x_5, \ldots, x_{n-3}, x_{n-1})$. When $x$ is a time series, $y$ is called the $2\times$ *down-sampled* version of $x$. If $x$ is a quantity sampled every hour, then $y$ is the same quantity, sampled every 2 hours.

**Image cropping.** As a more interesting example, suppose that $x$ is an image with $M \times N$ pixels, with $M$ and $N$ even. (That is, $x$ is an $MN$-vector, with its entries giving the pixels values in some specific order.) Let $y$ be the $(M/2) \times (N/2)$ image that is the upper left corner of the image $x$, *i.e.*, a cropped version. Then we have $y = Ax$, where $A$ is an $(MN)/4 \times (MN)$ selector matrix. The $i$th row of $A$ is $e_{k_i}^T$, where $k_i$ is the index of the pixel in $x$ that corresponds to the $i$th pixel in $y$.

**Permutation matrices.** An $n \times n$ *permutation matrix* is one in which each column is a unit vector, and each row is the transpose of a unit vector. (In other words, $A$ and $A^T$ are both selector matrices.) Thus, exactly one entry of each row is one, and exactly one entry of each column is one. This means that $y = Ax$ can be expressed as $y_i = x_{\pi_i}$, where $\pi$ is a permutation of $\{1, 2, \ldots, n\}$, *i.e.*, each integer from 1 to $n$ appears exactly once in $(\pi_1, \ldots, \pi_n)$.

As a simple example consider the permutation $\pi = (3, 1, 2)$. The associated permutation matrix is

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

Multiplying a 3-vector by $A$ re-orders its entries: $Ax = (x_3, x_1, x_2)$.

### 4.6.3 Incidence matrix

A *directed graph* consists of a set of *vertices* (or nodes), labeled $1, \ldots, n$, and a set of *directed edges* (or branches), labeled $1, \ldots, m$. Each edge is connected from one of the nodes and into another one. Directed graphs are often drawn with the vertices as circles or dots, and the edges as arrows, as in figure 4.3. A directed graph can be described by its $n \times m$ *incidence matrix*, defined as

$$A_{ij} = \begin{cases} 1 & \text{edge } j \text{ points to node } i \\ -1 & \text{edge } j \text{ points from node } i \\ 0 & \text{otherwise.} \end{cases}$$

The incidence matrix is evidently sparse, since it has only two nonzero entries in each column (one with value 1 and other with value $-1$). The $j$th column is

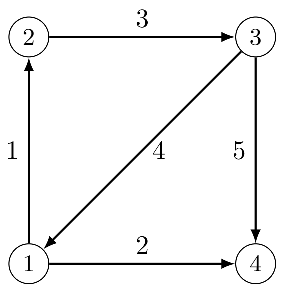

**Figure 4.3** Directed graph with four vertices and five edges.

associated with the $j$th edge; the indices of its two nonzero entries give the nodes that the edge connects. The $i$th row of $A$ corresponds to node $i$: its nonzero entries tell us which edges connect to the node, and whether they point into or away from the node. The incidence matrix for the graph shown in figure 4.3 is

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

The matrix-vector product $y = Ax$ can be given a very simple interpretation in terms of *flows* of some quantity along the edges of the graph. We interpret $x_j$ as the flow (rate) along the edge $j$, with a positive value meaning the flow is in the direction of edge $j$, and negative meaning the flow is in the opposite direction of edge $j$. The $n$-vector $y = Ax$ can be interpreted as the vector of net flows, from the edges, into the node: $y_i$ is equal to the total of the flows that come in to node $i$, minus the total of the flows that go out from node $i$.

If $Ax = 0$, we say that *flow conservation* occurs, since at each node, the total in-flow matches the total outflow. In this case the flow vector $x$ is called a *circulation.* This could be used as a model of traffic flow, with the nodes representing intersections and the edges representing road segments (one for each direction).

Another common equation is $Ax + s = 0$, where $s$ is an $n$-vector of *source flows*, which we can think of as an exogeneous flow that enters node $i$ (or leaves, when $s_i < 0$). The equation $Ax + s = 0$ means that the flow is conserved at each node: the total of all incoming flow from the in-coming edges and the source minus the total out-going flow from out-going edges is zero. This might be used as an approximate model of a power grid (ignoring losses), with $x$ being the vector of power flows along the transmission lines, $s_i > 0$ representing a generator injecting power into the grid at node $i$, $s_i < 0$ representing a load that consumes power at node $i$, and $s_i = 0$ representing a substation where power is exchanged among transmission lines, with no generation or load attached.

We can also give a simple interpretation to the matrix-vector product $u = A^T v$. Here $v$ is an $n$-vector, often interpreted as a *potential*, with $v_i$ the potential value at node $i$. The $m$-vector $u = A^T v$ gives the potential differences across the edges: $u_j = v_k - v_l$, where edge $j$ goes from node $k$ to node $l$.

The function of $v$ given by

$$\mathcal{L}(v) = \|A^T v\|^2$$

arises in many applications, and is called the *Laplacian* (associated with the graph). It can be expressed as

$$\mathcal{L}(v) = \sum_{\text{edges } (k,l)} (v_k - v_l)^2,$$

which is the sum of the squares of the differences of $v$ across all edges in the graph. The Laplacian is small when the potentials of nodes that are connnected by edges are near each other.

### 4.6.4 Convolution

The *convolution* of an $n$-vector $a$ and an $m$-vector $b$ is the $(n+m-1)$-vector denoted $c = a * b$, with entries

$$c_k = \sum_{i+j=k+1} a_i b_j, \quad k = 1, \ldots, n+m-1, \tag{4.4}$$

where the subscript in the sum means that we should sum over all values of $i$ and $j$ in their index ranges $1, \ldots, n$ and $1, \ldots, m$, for which the sum $i + j$ is $k + 1$. For example with $n = 4$, $m = 3$, we have

$$\begin{array}{rcl} c_1 & = & a_1 b_1 \\ c_2 & = & a_1 b_2 + a_2 b_1 \\ c_3 & = & a_1 b_3 + a_2 b_2 + a_3 b_1 \\ c_4 & = & a_2 b_3 + a_3 b_2 + a_4 b_1 \\ c_5 & = & a_3 b_3 + a_4 b_2 \\ c_6 & = & a_4 b_3. \end{array}$$

Convolution arises in many applications and contexts. For example, if $a$ and $b$ represent the coefficients of two polynomials,

$$p(x) = a_n x^{n-1} + \cdots + a_2 x + a_1, \qquad q(x) = b_n x^{m-1} + \cdots + b_2 x + b_1,$$

then the coefficients of the product polynomial $p(x)q(x)$ are represented by $c = a*b$:

$$p(x)q(x) = c_{n+m-1} x^{n+m-2} + \cdots + c_2 x + c_1.$$

Convolution is symmetric: we have $a * b = b * a$. It is also associative: we have $(a * b) * c = a * (b * c)$, so we can write both as $a * b * c$. (Both of these properties follow from the polynomial coefficient property above.) Another basic property is that for fixed $a$, the convolution $a * b$ is a linear function of $b$; and for fixed $b$, it is a linear function of $a$. This means we can express $a * b$ as a matrix-vector product:

$$a * b = T(b)a = T(a)b,$$

where $T(b)$ is the $(n+m-1)\times n$ matrix with entries

$$T(b)_{ij} = \begin{cases} b_{i-j+1} & 1 \le i-j+1 \le m \\ 0 & \text{otherwise.} \end{cases}$$

and similarly for $T(a)$. For example, with $n=4$ and $m=3$, we have

$$T(b) = \begin{bmatrix} b_1 & & & \\ b_2 & b_1 & & \\ b_3 & b_2 & b_1 & \\ & b_3 & b_2 & b_1 \\ & & b_3 & b_2 \\ & & & b_3 \end{bmatrix}, \qquad T(a) = \begin{bmatrix} a_1 & & \\ a_2 & a_1 & \\ a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 \\ & a_4 & a_3 \\ & & a_4 \end{bmatrix},$$

where entries not shown are zero. The matrices $T(b)$ and $T(a)$ are called *Toeplitz* matrices, since the entries any diagonal (*i.e.*, indices with $i-j$ constant) are the same. The columns of a Toeplitz matrix are simply shifted versions of a vector.

**Variations.** Several slightly different definitions of convolution are used in different applications. In one variation, $a$ and $b$ are infinite two-sided sequences (and not vectors) with indices ranging from $-\infty$ to $\infty$. In another variation, the rows of $T(a)$ at the top and bottom that do not contain all the coefficients of $a$ are dropped. (In this version, the rows of $T(a)$ are shifted versions of the vector $a$, reversed.) For consistency, we will use the one definition (4.4).

**Examples.**

- *Time series smoothing.* Suppose the $n$-vector $x$ is a time series, and $b = (1/3, 1/3, 1/3)$. Then $y = x * b$ can be interpreted as a *smoothed* version of the original time series: for $3 \le i \le n-3$, $y_i$ is the average of $x_i, x_{i-1}, x_{i-2}$. (We can drop the qualifier $3 \le i \le n-3$, by defining $x_i$ to be zero outside the index range $1, \ldots, n$.) The time series $y$ is called the (3-period) *moving average* of the time series $x$.

- *First order differences.* If the $n$-vector $x$ is a time series and $b = (-1, 1)$, the time series $y = x * b$ gives the first order differences in the series $x$: for $2 \le i \le n-2$, $y_i = x_i - x_{i-1}$.

- *Audio filtering.* If the $n$-vector $x$ is an audio signal, and $b$ is a vector (typically with length less than around 0.1 second of real time) the vector $y = x * b$ is called the *filtered* audio signal, with *filter coefficients* $b$. Depending on the coefficients $b$, $y$ will be perceived as enhancing or suppressing different frequencies, like the familiar audio tone controls.

- *Input-output system.* Many physical systems with an *input* (time series) $u$ and *output* time series $y$ are well modeled as $y = h * u$, where the vector $h$ is called the *system impulse response.* For example, $u_t$ might represent the power level of a heater at time period $t$, and $y_t$ might represent the resulting temperature rise (above the surrounding temperature).

- *Communication channel.* In a modern data communication system, a time series $x$ is transmitted or sent over some channel (*e.g.*, electrical, optical, or radio) to a receiver, which receives the time series $y$. A very common model is that $y$ and $u$ are related via convolution: $y = c * u$, where the vector $c$ is the *channel impulse response.*

**2-D convolution.** Convolution has a natural extension to mutliple dimensions. Suppose that $A$ is an $m \times n$ matrix, and $B$ is a $p \times q$ matrix. Their convolution is the $(n+p-1) \times (n+q-1)$ matrix

$$C_{rs} = \sum_{i+k=r+1,\ j+l=s+1} A_{ij} B_{kl}, \quad r = 1, \ldots, n+p-1, \quad s = 1, \ldots, m+q-1,$$

where the indices are restricted to their ranges (or alternatively, we assume that $A_{ij}$ and $B_{kl}$ are zero, when the indices are out of range). This in *not* denoted $C = A * B$, however, in standard mathematical notation. So we will use the notation $C = A \star B$.

The same properties that we observed for 1-D convolution hold for 2-D convolution: We have $A \star B = B \star A$, $(A \star B) \star C = A \star (B \star C)$, and for fixed $B$, $A \star B$ is a linear function.

**Image blurring.** If the $m \times n$ matrix $X$ represents an image, $Y = X \star B$ represents the effect of *blurring* the image by the *point spread function* (PSF) given by the entries of the matrix $B$. If we represent $X$ and $Y$ as vectors, we have $y = T(B)x$, for some $(m+p-1)(n+q-1) \times mn$-matrix $T(B)$.

As an example, with

$$B = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix},$$

$Y = X \star B$ is an image where each pixel value is the average of a $2 \times 2$ block of 4 adjacent pixels in $X$. The image $Y$ would be perceived as the image $X$, with some blurring of the fine details.

With the point spread function

$$D^{\text{hor}} = \begin{bmatrix} -1 & 1 \end{bmatrix},$$

the pixel values in the image $Y = X \star B$ are the horizontal first order differences of those in $X$:

$$Y_{i,j} = X_{i+1,j} - X_{i,j}, \quad i = 1, \ldots, n-1, \quad j = 1, \ldots, m.$$

With the point spread function

$$D^{\text{ver}} = \begin{bmatrix} -1 \\ 1 \end{bmatrix},$$

the pixel values in the image $Y = X \star B$ are the vertical first order differences of those in $X$:

$$Y_{i,j} = X_{i,j+1} - X_{i,j}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m-1.$$

# Chapter 5

# Linear equations

In this chapter consider vector-valued linear functions and systems of linear equations.

## 5.1 Linear and affine functions

**Vector valued functions of vectors.** The notation $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ means that $f$ is a function that maps real $n$-vectors to real $m$-vectors. 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 component $f_i$ of $f$ is itself a scalar valued function of $x$. As with 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. We use the same notation for each of the components of $f$, writing $f_i(x) = f_i(x_1, x_2, \ldots, x_n)$ to emphasize that $f_i$ is a function mapping the scalar arguments $x_1, \ldots, x_n$ into a scalar.

**The matrix-vector product function.** Suppose $A$ is an $m \times n$ matrix. We can define a function $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ by $f(x) = Ax$. Linear functions from $\mathbf{R}^n$ to $\mathbf{R}$, discussed in §1.7, are a special case with $m = 1$.

**Superposition and linearity.** The function $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$, defined by $f(x) = Ax$, is *linear*, *i.e.*, it satisfies the superposition property:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{5.1}$$

holds for all $n$-vectors $x$ and $y$ and all scalars $\alpha$ and $\beta$. It is a good exercise to parse this simple looking equation, since it involves overloading of notation. On the left-hand side, the scalar-vector multiplications $\alpha x$ and $\beta y$ involve $n$-vectors, and the sum $\alpha x + \beta y$ is the sum of two $n$-vectors. The function $f$ maps $n$-vectors to $m$-vectors, so $f(\alpha x + \beta y)$ is an $m$-vector. On the right-hand side, the scalar-vector multiplications and the sum are those for $m$-vectors. Finally, the equality sign is equality between two $m$-vectors.

We can verify that superposition holds for $f$ using properties of matrix-vector and scalar-vector multiplication:

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

Thus we can associate with every matrix $A$ a linear function $f(x)=Ax$.

The converse is also true. Suppose $f$ is a function that maps $n$-vectors to $m$-vectors, and is linear, *i.e.*, (5.1) holds for all $n$-vectors $x$ and $y$ and all scalars $\alpha$ and $\beta$. 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 §1.7, by showing that if $f$ is linear, then

$$f(x)=x_1f(e_1)+x_2f(e_2)+\cdots+x_nf(e_n), \tag{5.2}$$

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

$$A=\left[\begin{array}{cccc} f(e_1) & f(e_2) & \cdots & f(e_n)\end{array}\right].$$

The expression (5.2) 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 §1.7 it is easily shown that the matrix-vector representation of a linear function is unique. If $f:\mathbf{R}^n\rightarrow\mathbf{R}^m$ is a linear function, then there exists exactly one matrix $A$ such that $f(x)=Ax$ for all $x$.

**Examples.** Below we define six 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 $x$.

- *Negation.* $f$ changes the sign of $x$: $f(x)=-x$.

  The negation function is linear, because it can be expressed as $f(x)=Ax$ with $A=-I$.

- *Absolute value.* $f$ replaces each element of $x$ with its absolute value: $f(x)=(|x_1|,|x_2|,\ldots,|x_n|)$.

  The absolute value function is not linear. For example, with $n=1$, $x=1$, $y=0$, $\alpha=-1$, $\beta=0$, we have

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

  so superposition does not hold.

- *Reversal.* $f$ reverses the order of the elements of $x$: $f(x)=(x_n,x_{n-1},\ldots,x_1)$.

The reversal 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. It is sometimes called the *reverser matrix.*)

- *Sort.* $f$ sorts the elements of $x$ in decreasing order.

  The sort function is not linear (except when $n = 1$, in which case $f(x) = x$). For example, if $n = 2$, $x = (1, 0)$, $y = (0, 1)$, $\alpha = \beta = 1$, then

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

- *Running sum.* $f$ forms the running sum of the elements in $x$:

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

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

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

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

- *De-meaning.* $f$ subtracts the mean of a vector $x$: $f(x) = x - \mathbf{avg}(x)\mathbf{1}$.

  The de-meaning function is linear and can be expressed as $f(x) = Ax$ with

$$A = \begin{bmatrix} 1 - 1/n & -1/n & \cdots & -1/n \\ -1/n & 1 - 1/n & \cdots & -1/n \\ \vdots & \vdots & & \vdots \\ -1/n & -1/n & \cdots & 1 - 1/n \end{bmatrix}.$$

**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

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

holds for all $n$-vectors $x$, $y$, and all scalars $\alpha$, $\beta$ 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 $k$th unit vector in $\mathbf{R}^n$. We have

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

Just like affine scalar valued functions, affine vector valued functions are often called linear, even though they are linear only when the vector $b$ is zero.

## 5.2 Linear function models

Many functions or relations between variables that arise in natural science, engineering, and social sciences can be *approximated* as linear or affine functions. In these cases we refer to the linear function relating the two sets of variables as a *model* or an *approximation*, to remind us that the relation is only an approximation, and not exact. We give a few examples here.

- *Price-demand elasticity.* Consider $n$ goods or services with prices given by the $n$-vector $p$, and demands for the goods given by the $n$-vector $d$. A change in prices will induce a change in demands. We let $\delta^{\text{price}}$ be the $n$-vector that gives the fractional change in the prices, *i.e.*, $\delta_i^{\text{price}} = (p_i^{\text{new}} - p_i)/p_i$, where $p^{\text{new}}$ is the $n$-vector of new (changed) prices. We let $\delta^{\text{dem}}$ be the $n$-vector that gives the fractional change in the product demands, *i.e.*, $\delta_i^{\text{dem}} = (d_i^{\text{new}} - d_i)/d_i$, where $d^{\text{new}}$ is the $n$-vector of new demands. A linear elasticity model relates these vectors as $\delta^{\text{dem}} = E\delta^{\text{price}}$, where $E$ is the $n \times n$ *price elasticity matrix.* For example, suppose $E_{11} = -0.4$ and $E_{21} = 0.2$. This means that a 1% increase in the price of the first good, with other prices kept the same, will cause demand for the first good to drop by 0.4%, and demand for the second good to increase by 0.2%. (In this example, the second good is acting as a *partial substitute* for the first good.)

- *Elastic deformation.* Consider a steel structure like a bridge or the structural frame of a building. Let $f$ be an $n$-vector that gives the forces applied to the structure at $n$ specific places, sometimes called a *loading.* The structure will deform slightly due to the loading. Let $d$ be an $m$-vector that gives the displacement of various points in the structure due to the load, *e.g.*, the amount of sag at a specific point on a bridge. For small displacements, the relation between displacement and loading is very well approximated as linear: $d = Cf$, where $C$ is the $m \times n$ *compliance matrix.* The units of the entries of $C$ are given in m/N.

### 5.2.1 Taylor series approximation

Suppose $f : \mathbf{R}^n \to \mathbf{R}^m$ is differentiable, *i.e.*, has partial derivatives, and $z$ is an $n$-vector. The first-order Taylor approximation of $f$ near $z$ is given by

$$\begin{aligned}\hat{f}(x)_i &= f_i(z) + \frac{\partial f_i}{\partial x_1}(z)(x_1 - z_1) + \cdots + \frac{\partial f_i}{\partial x_n}(z)(x_n - z_n)\\ &= f_i(z) + \nabla f_i(z)^T(x - z),\end{aligned}$$

for $i = 1, \ldots, m$. (This is just the first-order Taylor approximation of each of the scalar-valued functions $f_i$.) For $x$ near $z$, $\hat{f}(x)$ is a very good approximation of $f(x)$. We can express this approximation in compact notation, using matrix-vector multiplication, as

$$\hat{f}(x) = f(z) + Df(z)(x - z),$$

where the $m \times n$ matrix $Df(z)$ is the *derivative* or *Jacobian* matrix of $f$ at $z$. Its components are the partial derivatives of $f$,

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

The rows of the Jacobian are $\nabla f_i(z)^T$, for $i = 1, \ldots, m$.

Evidently the Taylor series approximation $\hat{f}$ is an affine function of $x$. (It is often called a linear approximation of $f$, even though it is not, in general, a linear function.)

### 5.2.2 Regression model

Recall the regression model (1.7)

$$\hat{y} = x^T \beta + v,$$

where the $n$-vector $x$ is a feature vector for some object, $\beta$ is an $n$-vector of weights, $v$ is a constant (the offset), and $\hat{y}$ is the (scalar) value of the regression model response or outcome.

Now suppose we have a set of $N$ objects (also called *samples* or *examples*), with feature vectors $x_1, \ldots, x_N$. The regression model responses associated with the examples are given by

$$\hat{y}_i = x_i^T \beta + v, \quad i = 1, \ldots, N.$$

These numbers usually correspond to predictions of the value of the outputs or repsonses. If in addition to the example feature vectors $x_i$ we are also given the actual value of the associated response variables, $y_1, \ldots, y_N$, then our *prediction errors* are

$$\hat{y}_i - y_i, \quad i = 1, \ldots, N.$$

We can express this using compact matrix-vector notation. We form the $n \times N$ feature matrix $X$ with columns $x_1, \ldots, x_N$. We let $y$ denote the $N$-vector whose entries are the actual values of the response for the $N$ examples. We let $\hat{y}$ denote the $N$-vector of regression model responses for the $N$ examples. (So now $\hat{y}$ is an $N$-vector, whereas in the equations above, it represented a scalar.) We can then express the regression model for this data set in matrix-vector form as

$$\hat{y} = X^T \beta + v\mathbf{1}.$$

The vector of $N$ prediction errors for the examples is given by

$$\hat{y} - y = X^T \beta + v\mathbf{1} - y.$$

We can include the offset $v$ in the regression model by including an additional feature equal to one as the first entry of each feature vector:

$$\hat{y} = \left[\begin{array}{c} \mathbf{1}^T \\ X \end{array}\right]^T \left[\begin{array}{c} v \\ \beta \end{array}\right] = \tilde{X}^T \tilde{\beta},$$

where $\tilde{X}$ is the new feature matrix, with a new first row of ones, and $\tilde{\beta} = (v, \beta)$ is vector of regression model parameters. This is often written without the tildes, as $\hat{y} = X^T\beta$, by simply including the feature one as the first feature.

## 5.3 Linear dynamical systems

Suppose $x_1, x_2, \ldots$ is a sequence of $n$-vectors. The index (subscript) often denotes time or period, and is often written as $t$, so $x_t$ is the value of the sequence at time (or period) $t$. We can think of $x_t$ as a vector that changes over time, *i.e.*, one that changes dynamically. In this context, the sequence $x_1, x_2, \ldots$ is sometimes called a *trajectory*.

A *linear dynamical system* is a simple model for the sequence, in which each $x_{t+1}$ is a linear (or affine) function of $x_t$:

$$x_{t+1} = A_t x_t + b_t, \quad t = 1, 2, \ldots. \tag{5.3}$$

Here the $n \times n$ matrices $A_t$ are called the *dynamics matrices* and the $n$-vectors $b_t$ are called the offsets. The equation above is called the *dynamics* or *update* equation, since it gives us the next value of $x$, *i.e.*, $x_{t+1}$, as a function of the current value $x_t$. Often the dynamics matrices and offset vectors do not depend on $t$, in which case the linear dynamical system is called *time-invariant*.

If we know $x_t$ (and $A_t, A_{t+1}, \ldots$, $b_t, b_{t+1}, \ldots$) we can determine $x_{t+1}, x_{t+2}, \ldots$, simply by iterating the dynamics equation (5.3). In other words: If we know the current value of $x$, we can find all future values. For this reason it is called the *state* of the system; roughly speaking, it contains enough information to determine its future evolution.

There are many variations on and extensions of the basic linear dynamical system model (5.3), some of which we will encounter in the sequel. As an example, we can add an additional term to the update equation:

$$x_{t+1} = A_t x_t + B_t u_t + b_t, \quad t = 1, 2, \ldots.$$

Here $u_t$ is an $m$-vector called the *input* (at time $t$) and $B_t$ are $n \times m$ matices called the *input matrices*. The input is used to model other factors that affect the time evolution of the state.

**Population dynamics.** Linear dynamical systems can be used to describe the evolution of the age distribution in some population over time. Suppose $x_t$ is a 100-vector, with $(x_t)_i$ denoting the number of people in some population (say, a country) with age $i - 1$ (say, on January 1) in year $t$, where $t$ is measured starting from some base year, for $i = 1, \ldots, 100$. While $(x_t)_i$ is an integer, it is large enough that we simply consider it a real number. In any case, our model certainly is not accurate at the level of individual people. Also, note that the model does not track people 100 and older. The distribution of ages in the US in 2010 is shown in figure 5.1.

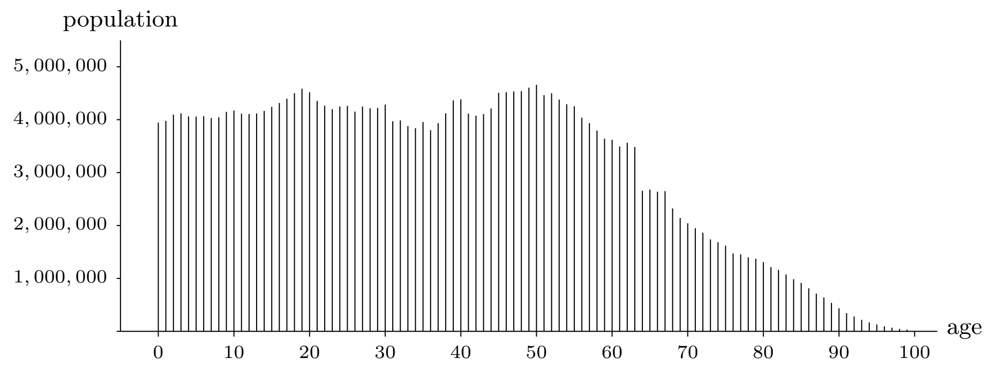

**Figure 5.1** Age distribution in the US in 2010. Source: United States Census Bureau (census.gov).

The birth rate is given by a 100-vector $b$, where $b_i$ is the average number of births per person with age $i-1$, $i = 1, \ldots, 100$. (This is half the average number of births per woman with age $i-1$, assuming equal numbers of men and women in the population.) Of course $b_i$ is approximately zero for $i < 13$ and $i > 50$. The approximate birth rates for the US in 2010 are shown in figure 5.2. The death rate is given by a 100-vector $d$, where $d_i$ is the portion of those aged $i-1$ who will die this year. The death rates for the US in 2010 are shown in figure 5.3.

To derive the dynamics equation, we find $x_{t+1}$ in terms of $x_t$, taking into account only births and deaths, and not immigration. The number of 0-year olds next year is the total number of births this year:

$$(x_{t+1})_1 = b^T x_t.$$

The number of $i$-year olds next year is the number of $(i-1)$-year-olds this year, minus those who die:

$$(x_{t+1})_{i+1} = (1-d_i)(x_t)_i, \quad i = 1, \ldots, 99.$$

We can assemble these equations into the simple linear dynamical system form

$$x_{t+1} = Ax_t, \quad t = 1, 2, \ldots,$$

where $A$ is given by

$$A = \begin{bmatrix} b_1 & b_2 & b_3 & \cdots & b_{98} & b_{99} & b_{100} \\ 1-d_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1-d_2 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1-d_{98} & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1-d_{99} & 0 \end{bmatrix}.$$

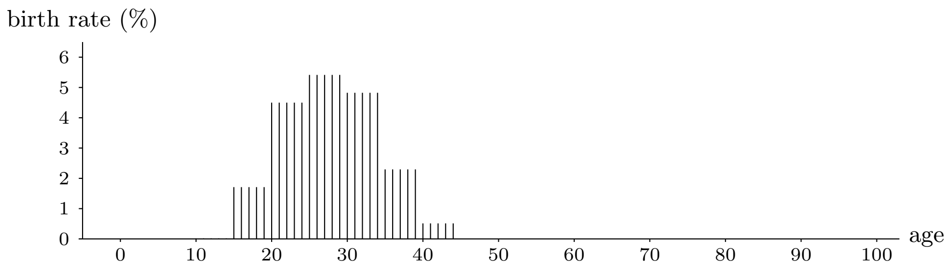

**Figure 5.2** Approximate birth rate versus age in the US in 2010. The figure is based on statistics for age groups of five years (hence, the piecewise-constant shape) and assumes an equal number of men and women in each age group. Source: Martin J.A., Hamilton B.E., Ventura S.J. *et al.*, Births: Final data for 2010. National Vital Statistics Reports; vol. 61, no. 1. National Center for Health Statistics (2012).

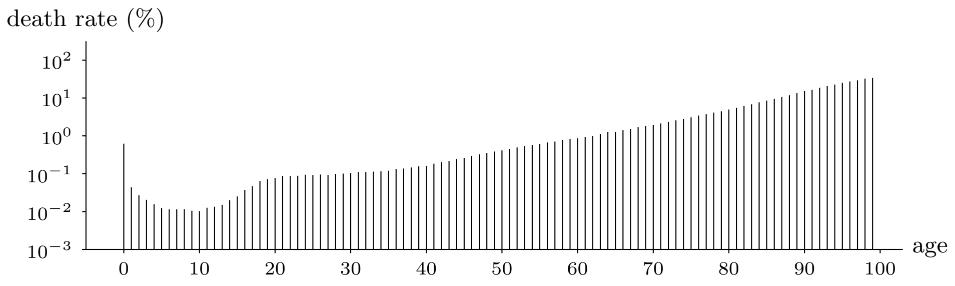

**Figure 5.3** Death rate versus age, for ages 0–99, in the US in 2010. Source: Centers for Disease Control and Prevention, National Center for Health Statistics (wonder.cdc.gov).

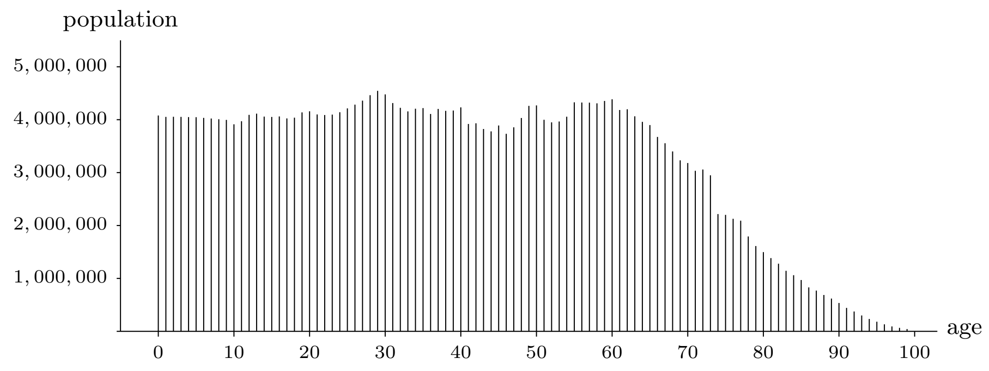

**Figure 5.4** Predicted age distribution in the US in 2020.

We can use this model to predict the total population in 10 years (not including immigration), or to predict the number of school age children, or retirement age adults. Figure 5.4 shows the predicted age distribution in 2020, computed by iterating the model $x_{t+1} = Ax_t$ for $t = 1, \ldots, 10$, with initial value $x_1$ given by the 2010 age distribution of figure 5.1. Note that the distribution is based on an approximate model, since we neglect the effect of immigration, and assume that the death and birth rates remain constant and equal to the values shown in figures 5.2 and 5.3.

## 5.4 Systems of linear equations

Consider a set (also called a system) of $m$ linear equations in $n$ variables or unknowns $x_1, \ldots, x_n$:

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

The numbers $A_{ij}$ are called the *coefficients* in the linear equations, and the numbers $b_i$ are called the *right-hand sides* (since by tradition, they appear on the right-hand side of the equation). These equations can be written succinctly in matrix notation as

$$Ax = b. \tag{5.4}$$

In this context, the $m \times n$ matrix $A$ is called the *coefficient matrix*, and the $m$-vector $b$ is called the *right-hand side*. An $n$-vector $x$ is called a *solution* of the linear equations if $Ax = b$ holds. A set of linear equations can have no solutions, one solution, or multiple solutions.

The set of linear equations is called *over-determined* if $m > n$, *under-determined* if $m < n$, and *square* if $m = n$; these correspond to the coefficient matrix being tall, wide, and square, respectively. When the system of linear equations is over-determined, there are more equations than variables or unknowns. When the system of linear equations is under-determined, there are more unknowns than equations. When the system of linear equations is square, the numbers of unknowns and equations is the same. A set of equations with zero right-hand side, $Ax = 0$, is called a *homogeneous* set of equations. Any homogeneous set of equations has $x = 0$ as a solution.

In chapter 7 we will address the question of how to determine if a system of linear equations has a solution, and how to find one when it does. For now, we give a few interesting examples.

### 5.4.1 Examples

**Coefficients of linear combinations.** Let $a_1, \ldots, a_n$ denote the columns of $A$. The system of linear equations $Ax = b$ can be expressed as

$$x_1 a_1 + \cdots + x_n a_n = b,$$

*i.e.*, $b$ is a linear combination of $a_1, \ldots, a_n$ with coefficients $x_1, \ldots, x_n$. So solving $Ax = b$ is the same as finding coefficients that express $b$ as a linear combination of the vectors $a_1, \ldots, a_n$.

**Polynomial interpolation.** We seek a polynomial $p$ of degree at most $n - 1$ that interpolates a set of $m$ given points $(t_i, y_i)$, $i = 1, \ldots, m$. (This means that $p(t_i) = y_i$.) We can express this as a set of $m$ linear equations in the $n$ unknowns $c$, where $c$ is the $n$-vector of coefficients: $Ac = y$. Here the matrix $A$ is the Vandermonde matrix (4.3), and the vector $c$ is the vector of polynomial coefficietns, as described in the example on page 63.

**Balancing chemical reactions.** A chemical reaction involves $p$ reactants (molecules) and $q$ products, and can be written as

$$a_1 R_1 + \cdots + a_p R_p \longrightarrow b_1 P_1 + \cdots + b_q P_q.$$

Here $R_1, \ldots, R_p$ are the reactants, $P_1, \ldots, P_q$ are the products, and the numbers $a_1, \ldots, a_p$ and $b_1, \ldots, b_q$ are positive numbers that tell us how many of each of these molecules is involved in the reaction. They are typically integers, but can be scaled arbitrarily; we could double all of these numbers, for example, and we still have the same reaction. As a simple example, we have the electrolysis of water,

$$2\mathrm{H_2O} \longrightarrow 2\mathrm{H_2} + \mathrm{O_2},$$

which has one reactant, water ($H_2O$) and two products: molecular hydrogen ($H_2$), and molecular oxygen ($O_2$). The coefficients tell us that 2 water molecules create 2 hydrogen molecules and 1 oxygen molecule.

In a chemical reaction the numbers of constituent atoms must balance. This means that for each atom appearing in any of the reactants or products, the total amount on the left-hand side must equal the total amount on the right-hand side. (If any of the reactants or products is charged, *i.e.*, an ion, then the total charge must also balance.) In the simple water electrolysis reaction above, for example, we have 4 hydrogen atoms on the left (2 water molecules, each with hydrogen atoms), and 4 on the right (2 hydrogen molecules, each with 2 hydrogen atoms). The oxygen atoms also balance, so this reaction is balanced.

We can express the requirement that the reaction balances as a set of $m$ equations, where $m$ is the number of different atoms appearing in the chemical reaction. We define the $m \times p$ matrix $R$ by

$$R_{ij} = \text{number of atoms of type } i \text{ in } R_j, \quad i = 1, \ldots, m, \quad j = 1, \ldots, p.$$

(The entries of $R$ are nonnegative integers.) The matrix $R$ is interesting; for example, its $j$th column gives the chemical formula for reactant $R_j$. We let $a$ denote the $p$-vector with entries $a_1, \ldots, a_p$. Then, the $m$-vector $Ra$ gives the total number of atoms of each type appearing in the reactants.

We define an $m \times q$ matrix $P$ in a similar way, so the $m$-vector $Pb$ gives the total number of atoms of each type that appears in the products.

We write the balance condition using vectors and matrices as $Ra = Pb$. We can express this as

$$\left[ \begin{array}{cc} R & -P \end{array} \right] \left[ \begin{array}{c} a \\ b \end{array} \right] = 0,$$

which is a set of $m$ homogeneous linear equations.

A simple solution of these equations is $a = 0$, $b = 0$. But we seek a nonzero solution. We can set one of the coefficients, say $a_1$, to be one. (This might cause the other quantities to be fractional valued.) We can add the condition that $a_1 = 1$ to our system of linear equations as

$$\left[ \begin{array}{cc} R & -P \\ e_1^T & 0 \end{array} \right] \left[ \begin{array}{c} a \\ b \end{array} \right] = e_{m+1}.$$

Finally, we have a set of $m + 1$ equations in $p + q$ variables that expresses the requirement that the chemical reaction balances. Finding a solution of this set of equations is called *balancing* the chemical reaction.

**Diffusion systems.** A *diffusion system* is a common model that arises in many areas of physics to describe *flows* and *potentials*. We start with a directed graph with $n$ nodes and $m$ edges. Some quantity (like electricity, heat, energy, or mass) can flow across the edges, from one node to another.

With edge $j$ we associate a flow (rate) $f_j$, which is a scalar; the vector of all $m$ flows is the flow $m$-vector $f$. The flows $f_j$ can be positive or negative: Positive $f_j$ means the quantity flows in the direction of edge $j$, and negative $f_j$ means the quantity flows in the opposite direction of edge $j$. The flows can represent, for example, heat flow (in units of Watts) in a thermal model, electrical current (Amps) in an electrical circuit, or movement (diffusion) of mass (such as, for example, a pollutant). We also have a source (or exogenous) flow $s_i$ at each node, with $s_i > 0$

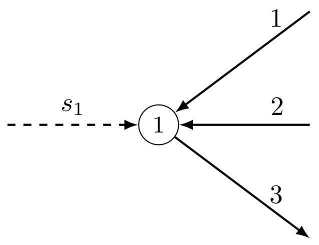

**Figure 5.5** A node in a diffusion system with label 1, exogeneous flow $s_1$ and three incident edges.

meaning that an exogenous flow is injected into node $i$, and $s_i < 0$ means that an exogenous flow is removed from node $i$. (In some contexts, a node where flow is removed is called a *sink*.) In a thermal system, the sources represent thermal (heat) sources; in an electrical circuit, they represent electrical current sources; in a system with diffusion, they represent external injection or removal of the mass.

In a diffusion system, the flows must satisfy (flow) *conservation*, which means that at each node, the total flow entering each node from adjacent edges and the exogenous source, must be zero. This is illustrated in figure 5.5, which shows three edges adjacent to node 1, two entering node 1 (flows 1 and 2), and one (flow 3) leaving node 1, and an exogeneous flow. Flow conservation at this node is expressed as

$$f_1 + f_2 - f_3 + s_1 = 0.$$

Flow conservation at every node can be expressed by the simple matrix-vector equation

$$Af + s = 0, \tag{5.5}$$

where $A$ is the incidence matrix described in §4.6.3. (This is called *Kirchhoff's current law* in an electrical circuit; when the flows represent movement of mass, it is called *conservation of mass.*)

With node $i$ we associate a potential $e_i$; the $n$-vector $e$ gives the potential at all nodes. The potential might represent the node temperature in a thermal model, the electrical potential (voltage) in an electrical circuit, and the concentration in a system that involves mass diffusion.

In a diffusion system *the flow on an edge is proportional to the potential difference across its adjacent nodes*. This is typically written as $r_j f_j = e_k - e_l$, where edge $j$ goes from node $k$ to node $l$, and $r_j$ (which is typically positive) is called the *resistance* of edge $j$. In a thermal model, $r_j$ is called the thermal resistance of the edge; in a electrical circuit, it is called the electrical resistance. This is illustrated in figure 5.6, which shows edge 8, connecting node 2 and node 3, corresponding to an edge flow equation

$$r_8 f_8 = e_2 - e_3.$$

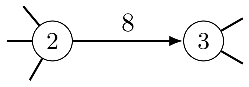

**Figure 5.6** The flow through edge 8 is equal to $f_8 = r_8(e_2 - e_3)$.

We can write the edge flow equations in a compact way as

$$Rf = -A^T e, \tag{5.6}$$

where $R = \mathbf{diag}(r)$ is called the *resistance matrix*.

The diffusion model can be expressed as one set of block linear equations in the variables $f$, $s$, and $e$:

$$\left[\begin{array}{ccc} A & I & 0 \\ R & 0 & A^T \end{array}\right] \left[\begin{array}{c} f \\ s \\ e \end{array}\right] = 0.$$

This is a set of $n+m$ homogeneous equations in $m+2n$ variables. To these underdetermined equations we can add others, for example, by specifying some of the entries of $f$, $s$, and $e$.

Chapter 6

# Matrix multiplication

In this chapter we introduce matrix multiplication, and describe several applications.

## 6.1 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

$$C_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj} = A_{i1}B_{1j} + \cdots + A_{ip}B_{pj}, \qquad i = 1,\ldots,m, \quad j = 1,\ldots,n. \quad (6.1)$$

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

As a specific example, we have

$$\left[\begin{array}{rrr} -1.5 & 3 & 2 \\ 1 & -1 & 0 \end{array}\right] \left[\begin{array}{rr} -1 & -1 \\ 0 & -2 \\ 1 & 0 \end{array}\right] = \left[\begin{array}{rr} 3.5 & -4.5 \\ -1 & 1 \end{array}\right].$$

To find the $1,2$ entry of the right-hand matrix, we move along the first row of the left-hand matrix, and down the second column of the middle matrix, to get $(-1.5)(-1) + (3)(-2) + (2)(0) = -4.5$.

Matrix-matrix multiplication includes as special cases several other types of multiplication (or product) we have encountered so far.

**Scalar-vector product.** If $x$ is a scalar and $a$ is a number, we can interpret the scalar-vector product $xa$, with the scalar appearing on the right, as a special case of matrix-matrix multiplication. We consider $x$ as an $n \times 1$ matrix, and $a$ as a $1 \times 1$ matrix. The matrix product $xa$ then makes sense, and coincides with the scalar-vector product $xa$, which we usually write (by convention) as $ax$. But note that $ax$ cannot be interpreted as matrix-matrix multiplication (except when $n = 1$), since the number of columns of $a$ (which is 1) is not equal to the number of rows of $x$ (which is $n$).

**Inner product.** An important special case of matrix-matrix multiplication is the multiplication of a row vector with a column vector. If $a$ and $b$ are $n$-vectors, then the inner product

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

can be interpreted as the matrix-matrix product of the $1 \times n$ matrix $a^T$ and the $n \times 1$ matrix $b$. The result is a $1 \times 1$ matrix, which we consider to be a scalar. (This explains the notation $a^T b$ for the inner product of vectors $a$ and $b$, defined in §1.6.)

**Matrix-vector multiplication.** The matrix-vector product $y = Ax$ defined in (4.2) can be interpreted as a matrix-matrix product of $A$ with the $n \times 1$ matrix $x$.

**Vector outer product.** The *outer product* of an $m$-vector $a$ and an $n$-vector $b$ is given by $ab^T$, which is an $m \times n$ matrix

$$ab^T = \left[\begin{array}{cccc} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & & \vdots \\ a_m b_1 & a_m b_2 & \cdots & a_m b_n \end{array}\right],$$

whose entries are all products of the entries of $a$ and the entries of $b$. Note that the outer product does not satisfy $ab^T = ba^T$, *i.e.*, it is not symmetric (like the inner product). Indeed, the equation $ab^T = ba^T$ does not even make sense, unless $m = n$; even then, it is not true in general.

**Properties of matrix multiplication.** 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 $2 \times 3$ and $B$ is $3 \times 4$, then $AB$ makes sense (the dimensions are compatible) but $BA$ does not even make sense (the dimensions are incompatible). 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

$$A = \left[\begin{array}{cc} 1 & 6 \\ 9 & 3 \end{array}\right], \qquad B = \left[\begin{array}{rr} 0 & -1 \\ -1 & 2 \end{array}\right].$$

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*. (Note that for $AB = BA$ to make sense, $A$ and $B$ must both be square.)

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

- *Associativity:* $(AB)C = A(BC)$. Therefore we write the product simply as $ABC$.
- *Associativity with scalar multiplication:* $\gamma(AB) = (\gamma A)B$, where $\gamma$ 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 $\gamma A$ and $\gamma B$ are defined as scalar-matrix products, but in general, unless $A$ and $B$ have one row, not as matrix-matrix products.)
- *Distributivity with addition:* $A(B+C) = AB+AC$ and $(A+B)C = AC+BC$.
- *Transpose of product.* The transpose of a product is the product of the transposes, but in the *opposite* order: $(AB)^T = B^TA^T$.

**Inner product and matrix-vector products.** 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 $A^Ty$.

**Products of block matrices.** Suppose $A$ is a block matrix with $m \times p$ block entries $A_{ij}$, and $B$ is a block matrix with $p \times q$ block entries $B_{ij}$, and for each $k = 1, \ldots, p$, the matrix product $A_{ik}B_{kj}$ makes sense (*i.e.*, the number of columns of $A_{ik}$ equals the number of rows of $B_{kj}$). Then $C = AB$ can be expressed as the $m \times n$ block matrix with entries $C_{ij}$, given by the formula (6.1). For example, we have

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix},$$

for any matrices $A, B, \ldots, H$ for which the matrix products above make sense.

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

Consider the matrix product of an $m \times p$ matrix $A$ and an $p \times n$ 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

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

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 $A$.

**Inner product representation.** From the definition of the $i, j$ element of $AB$ in (6.1), we also see that the elements of $AB$ are the inner products of the rows of $A$ with the columns of $B$:

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

Thus we can interpret the matrix-matrix product as the $mn$ inner products $a_i^T b_j$ arranged in an $m \times n$ matrix.

**Gram matrix.** For an $m \times n$ matrix $A$, with columns $a_1, \ldots, a_n$, the matrix product $G = A^T A$ is called the *Gram matrix* associated with the set of $m$-vectors $a_1, \ldots, a_n$. From the iner product interpretation above, the Gram matrix can be expressed as

$$G = A^T A = \begin{bmatrix} a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\ a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n^T a_1 & a_n^T a_2 & \cdots & a_n^T a_n \end{bmatrix}.$$

The entries of the Gram matrix $G$ give all inner products of pairs of columns of $A$. Note that a Gram matrix is symmetric, since $a_i^T a_j = a_j^T a_i$. This can also be seen using the transpose of product rule:

$$G^T = (A^T A)^T = (A^T)(A^T)^T = A^T A = G.$$

**Outer product representation.** If we express the $m \times p$ matrix $A$ in terms of its columns $a_1, \ldots, a_p$ and the $p \times n$ matrix $B$ in terms of its rows $b_1^T, \ldots, b_p^T$,

$$A = \begin{bmatrix} a_1 & \cdots & a_p \end{bmatrix}, \qquad B = \begin{bmatrix} b_1^T \\ \vdots \\ b_p^T \end{bmatrix},$$

then we can express the product matrix $AB$ as

$$AB = a_1 b_1^T + \cdots + a_p b_p^T,$$

a sum of outer products.

**Complexity of matrix multiplication.** The total number of flops required for a matrix-matrix product $C = AB$ with $A$ of size $m \times p$ and $B$ of size $p \times n$ can be found several ways. The product matrix $C$ has size $m \times n$, so there are $mn$ elements to compute. The $i, j$ element of $C$ is the inner product of row $i$ of $A$ with column $j$ of $B$. This is an inner product of vectors of length $p$ and requires $2p - 1$ flops. Therefore the total is $mn(2p - 1)$ flops, which we approximate as $2mnp$ flops. The order of computing the matrix-matrix product is $mnp$, the product of the three dimensions involved.

In some special cases the complexity is less than $2mnp$ flops. As an example, when we compute the $m \times m$ Gram matrix $G = A^T A$ we only need to compute the entries in the upper (or lower) half of $G$, since $G$ is symmetric. This saves around half the flops, so the complexity is around $mn^2$ flops. But the order is the same.

**Complexity of matrix triple product.** Consider the product of three matrices,

$$D = ABC$$

with $A$ of size $m \times n$, $B$ of size $n \times p$, and $C$ of size $p \times q$. The matrix $D$ can be computed in two ways, as $(AB)C$ and as $A(BC)$. In the first method we start with $AB$ ($2mnp$ flops) and then form $D = (AB)C$ ($2mpq$ flops), for a total of $2mp(n+q)$ flops. In the second method we compute the product $BC$ ($2npq$ flops) and then form $D = A(BC)$ ($2mnq$ flops), for a total of $2nq(m + p)$ flops.

You might guess that the total number of flops required is the same with the two methods, but it turns out it is not. The first method is less expensive when $2mp(n + q) < 2nq(m + p)$, *i.e.*, when

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

For example, if $m = p$ and $n = q$, the first method has a complexity proportional to $m^2 n$, while the second method has complexity $mn^2$, and one would prefer the first method when $m \ll n$.

## 6.2 Composition of linear functions

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

$$h(x) = f(g(x)) = A(Bx) = (AB)x.$$

In words: to find $h(x)$, we first apply the function $g$, to obtain the partial result $g(x)$ (which is a $p$-vector); then we apply the function $f$ to this result, to obtain $h(x)$ (which is an $m$-vector). In the formula $h(x) = f(g(x))$, $f$ appears to the left of $g$; but when we evaluate $h(x)$, we apply $g$ first. The composition $h$ is evidently a linear function, that can be written as $h(x) = Cx$ with $C = AB$.

Using this interpretation it is easy to understand 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 $2 \times 2$ 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 $f(g(x)) = ABx = (-x_2, x_1)$, we first reverse the order, and then change the sign of the first element. This result is obviously different from $g(f(x)) = BAx = (x_2, -x_1)$, obtained by changing the sign of the first element, and then reversing the order of the elements.

**Composition of affine functions.** The composition of affine functions is an affine function. Suppose $f : \mathbf{R}^p \to \mathbf{R}^m$ is the affine function given by $f(x) = Ax + b$, and $g : \mathbf{R}^n \to \mathbf{R}^p$ is the affine function given by $g(x) = Cx + d$. The composition $h$ is given by

$$h(x) = f(g(x)) = A(Cx + d) + b = (AC)x + (Ad + b) = \tilde{A}x + \tilde{b},$$

where $\tilde{A} = AC$, $\tilde{b} = Ad + b$.

**Chain rule of differentiation.** Let $f : \mathbf{R}^p \to \mathbf{R}^m$ and $g : \mathbf{R}^n \to \mathbf{R}^p$ be differentiable functions. The composition of $f$ and $g$ is defined as the function $h : \mathbf{R}^n \to \mathbf{R}^m$ with

$$h(x) = f(g(x)) = f(g_1(x), \ldots, g_p(x)).$$

The function $h$ is differentiable and its partial derivatives follow from those of $f$ and $g$ via the chain rule:

$$\frac{\partial h_i}{\partial x_j}(z) = \frac{\partial f_i}{\partial y_1}(g_1(z))\frac{\partial g_1}{\partial x_i}(z) + \cdots + \frac{\partial f_i}{\partial y_p}(g_p(z))\frac{\partial g_p}{\partial x_i}(z)$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. This relation can be expressed concisely as a matrix-matrix product: the derivative matrix of $h$ at $z$ is the product

$$Dh(z) = Df(g(z))Dg(z)$$

of the derivative matrix of $f$ at $g(z)$ and the derivative matrix of $g$ at $z$. This compact matrix formula generalizes the chain rule for scalar-valued functions of a single variable, *i.e.*, $h'(z) = f'(g(z))g'(z)$.

The first order Taylor approximation of $h$ at $z$ can therefore be written as

$$\begin{aligned}\hat{h}(x) &= h(z) + Dh(z)(x-z) \\ &= f(g(z)) + Df(g(z))Dg(z)(x-z).\end{aligned}$$

The same result can be interpreted as a composition of two affine functions, the first order Taylor approximation of $f$ at $g(z)$,

$$\hat{f}(y) = f(g(z)) + Df(g(z))(y - g(z))$$

and the first order Taylor approximation of $g$ at $z$,

$$\hat{g}(x) = g(z) + Dg(z)(x-z).$$

The composition of these two affine functions is

$$\begin{aligned}\hat{f}(\hat{g}(x)) &= \hat{f}(g(z) + Dg(z)(x-z)) \\ &= f(g(z)) + Df(g(z))(g(z) + Dg(z)(x-z) - g(z)) \\ &= f(g(z)) + Df(g(z))Dg(z)(x-z)\end{aligned}$$

which is equal to $\hat{h}(x)$.

When $h$ and $f$ are a scalar valued function ($m = 1$), the derivative matrices $Dh(z)$ and $Df(g(z))$ are the transposes of the gradients, and we write the chain rule as

$$\nabla h(z) = Dg(z)^T \nabla f(g(z)).$$

In particular, if $g(x) = Ax + b$ is affine, then the gradient of $h(x) = f(g(x)) = f(Ax+b)$ is given by $\nabla h(z) = A^T \nabla f(Ax+b)$.

## 6.3 Matrix power

**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 $(A^k)^l = A^{kl}$. By convention we take $A^0 = I$, which makes the formulas above hold for nonnegative integer values of $k$ and $l$.

Matrix powers $A^k$ with $k$ a negative integer are discussed in §7.2. 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 that we will not pursue in this book.

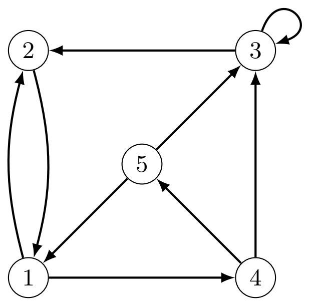

**Figure 6.1** Directed graph.

**Paths in directed graph.** Suppose $A$ is an $n \times n$ matrix representing a directed graph with $n$ vertices:

$$A_{ij} = \begin{cases} 1 & \text{there is a edge from vertex } j \text{ to vertex } i \\ 0 & \text{otherwise} \end{cases}$$

(or a relation $\mathcal{R}$ on $\{1, 2, \ldots, n\}$; see page 57). The elements of the matrix powers $A^\ell$ have a simple meaning in terms of directed paths in the graph. First examine the expression for the $i, j$ element of the square of $A$:

$$(A^2)_{ij} = \sum_{k=1}^{n} A_{ik} A_{kj}.$$

Each term in the sum is 0 or 1, and equal to one only if there is an edge from vertex $j$ to vertex $k$ and an edge from vertex $k$ to vertex $i$, *i.e.*, a directed path of length exactly two from vertex $j$ to vertex $i$ via vertex $k$. By summing over all $k$, we obtain the total number of paths of length two from $j$ to $i$. The matrix $A$ representing the graph in figure 6.1, for example, and its square are given by

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \qquad A^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

We can verify there is exactly one directed path of length two from vertex 1 to itself, *i.e.*, the path $(1, 2, 1)$), and one directed path of length two from vertex 3 to vertex 1, *i.e.*, the path $(3, 2, 1)$. There are two paths of length two from vertex 4 to vertex 3: the paths $(4, 3, 3)$ and $(4, 5, 3)$, so $A_{34} = 2$.

The property extends to higher powers of $A$. If $\ell$ is a positive integer, then the $i, j$ element of $A^\ell$ is the number of directed paths of length $\ell$ from vertex $j$ to vertex $i$. This can be proved by induction on $\ell$. We have already shown the result for $\ell = 2$. Assume that it is true that the elements of $A^\ell$ give the paths of length $\ell$ between the different vertices. Consider the expression for the $i, j$ element of $A^{\ell+1}$:

$$(A^{\ell+1})_{ij} = \sum_{k=1}^{n} A_{ik} (A^\ell)_{kj}.$$

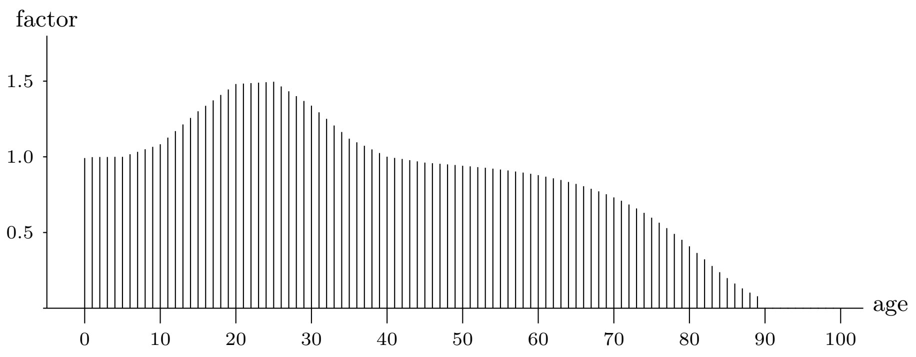

**Figure 6.2** Contribution factor per age in 2010 to the total population in 2020. The value for age $i-1$ is the $i$th component of the row vector $\mathbf{1}^T A^{10}$.

The $k$th term in the sum is equal to the number of paths of length $\ell$ from $j$ to $k$ if there is an edge from $k$ to $i$, and is equal to zero otherwise. Therefore it is equal to the number of paths of length $\ell+1$ from $j$ to $i$ that end with the edge $(k,i)$, *i.e.*, of the form $(j,\ldots,k,i)$. By summing over all $k$ we obtain the total number of paths of length $\ell+1$ from vertex $j$ to $i$. This can be verified in the example. The third power of $A$ is

$$A^3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 \\ 2 & 0 & 2 & 3 & 1 \\ 2 & 1 & 1 & 2 & 2 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

The $(A^3)_{24} = 3$ paths of length three from vertex 4 to vertex 2 are $(4,3,3,2)$, $(4,5,3,2)$, $(4,5,1,2)$.

**Linear dynamical systems.** Consider a time-invariant linear dynamical system, described by $x_{t+1} = Ax_t$. We have $x_{t+2} = Ax_{t+1} = A(Ax_t) = A^2 x_t$. Continuing this argument, we have

$$x_{t+\ell} = A^\ell x_t,$$

for $\ell = 1, 2, \ldots$. In a linear dynamical system, we can interpret $A^\ell$ as the matrix that propagates the state forward $\ell$ steps. For example, in a population dynamics model, $A^\ell$ is the matrix that maps the current population distribution into the population distribution $\ell$ periods in the future, taking into account births, deaths, and the births and deaths of children, and so on. The total population $\ell$ periods in the future is given by $\mathbf{1}^T(A^\ell x_t)$, which we can write as $(\mathbf{1}^T A^\ell)x_t$. The row vector $\mathbf{1}^T A^\ell$ has an interesting interpretation: its $i$th entry is the contribution to the total population in $\ell$ periods due to each person with current age $i-1$. It is plotted in figure 6.2 for the US data given above.

## 6.4 QR factorization

**Matrices with orthonormal columns.** As an application of Gram matrices, we can express the condition that the $n$-vectors $a_1, \ldots, a_k$ are orthonormal in a simple way using matrix notation:

$$A^T A = I,$$

where $A$ is the $n \times k$ matrix with columns $a_1, \ldots, a_k$. There is no standard term for a matrix whose columns are orthonormal: We refer to a matrix whose columns are orthonormal as 'a matrix whose columns are orthonormal'. But a *square* matrix that satisfies $A^T A = I$ is called *orthogonal*; its columns are an orthonormal basis. Orthogonal matrices have many uses, and arise in many applications.

We have already encountered some orthogonal matrices, including identity matrices, 2-D reflections and rotations (page 65), and permutation matrices (page 66).

**Norm, inner product, and angle properties.** Suppose the columns of the $m \times n$ matrix $A$ are orthonormal, and $x$ and $y$ are any $n$-vectors. We let $f : \mathbf{R}^n \to \mathbf{R}^m$ be the function that maps $z$ to $Az$. Then we have the following:

- $\|Ax\| = \|x\|$. That is, $f$ is *norm preserving.*
- $(Ax)^T(Ay) = x^T y$. $f$ preserves the inner product between vectors.
- $\angle(Ax, Ay) = \angle(x, y)$. $f$ also preserves angles between vectors.

Note that in each of the three equations above, the vectors appearing in the left- and right-hand sides have different dimensions, $m$ on the left and $n$ on the right.

We can verify these properties using simple matrix properties. We start with the second statement, that multiplication by $A$ preserves the inner product. We have

$$\begin{aligned} (Ax)^T(Ay) &= (x^T A^T)(Ay) \\ &= x^T(A^T A)y \\ &= x^T I y \\ &= x^T y. \end{aligned}$$

In the first line, we use the transpose-of-product rule; in the second, we re-associate a product of 4 matrices (considering the row vector $x^T$ and column vector $x$ as matrices); in the third line we use $A^T A = I$, and in the fourth line we use $Iy = y$.

From the second property we can derive the first one: By taking $y = x$ we get $(Ax)^T(Ax) = x^T x$; taking the squareroot of each side gives $\|Ax\| = \|x\|$. The third property, angle preservation, follows from the first two, since

$$\angle(Ax, Ay) = \arccos\left(\frac{(Ax)^T(Ay)}{\|Ax\|\|Ay\|}\right) = \arccos\left(\frac{x^T y}{\|x\|\|y\|}\right) = \angle(x, y).$$

**QR factorization.** We can express the Gram-Schmidt factorization described in §3.4 in a compact form using matrices. Let $A$ be an $n \times k$ matrix with linearly independent columns $a_1, \ldots, a_k$. By the independence-dimension inequality, $A$ is tall or square.

Let $Q$ be the $n \times k$ matrix with columns $q_1, \ldots, q_k$, the orthonormal vectors produced by the Gram-Schmidt algorithm applied to the $n$-vectors $a_1, \ldots, a_k$. Orthonormality of $q_1, \ldots, q_k$ is expressed in matrix form as $Q^T Q = I$.

We express the equation relating $a_i$ and $q_i$,

$$a_i = (q_1^T a_i) q_1 + \cdots + (q_{i-1}^T a_i) q_{i-1} + \|\tilde{q}_i\| q_i,$$

where $\tilde{q}_i$ is the vector obtained in the first step of the Gram-Schmidt algorithm, as

$$a_i = R_{1i} q_1 + \cdots + R_{ii} q_i,$$

where $R_{ij} = q_i^T a_j$ for $i < j$ and $R_{ii} = \|\tilde{q}_i\|$. Defining $R_{ij} = 0$ for $i > j$, we can express the equations above in compact matrix form as

$$A = QR.$$

This is called the *QR factorization* of $A$, since it expresses the matrix $A$ as a product of two matrices, $Q$ and $R$. The $n \times k$ matrix $Q$ has orthonormal columns, and the $k \times k$ matrix $R$ is upper triangular, with positive diagonal elements. If $A$ is square of order $n$, with linearly independent columns, then $Q$ is orthogonal and the QR factorization expresses $A$ as a product of two square matrices.

The Gram-Schmidt algorithm is not the only algorithm for QR factorization. Several other QR factorization algorithms exist, that are more reliable in the presence of round-off errors. There also exist variants that efficiently handle the case when the the matrix $A$ is sparse. (These QR factorization methods extensions also change the *order* in which the columns of $A$ are processed.) In this case the matrix $Q$ is typically also sparse, so storing it requires (often much) less memory than if they were dense (*i.e.*, $nk$ numbers). The flop count is also (often much) smaller than $nk^2$.

# Chapter 7

# Matrix inverses

In this chapter we introduce the concepts of matrix inverses. We show how they can be used to solve linear equations, and how they can be computed using the QR factorization.

## 7.1 Left and right inverses

Recall that for a number $a$, its (multiplicative) inverse is the number $x$ for which $xa = 1$, which we usually denote as $x = 1/a$ or (less frequently) $x = a^{-1}$. The inverse $x$ exists provided $a$ is nonzero. For matrices the concept of inverse is more complicated than for scalars; in the general case, we need to distinguish between left and right inverses. We start with the left inverse.

**Left inverse.** A matrix $X$ that satisfies

$$XA = I$$

is called a *left inverse* of $A$. The matrix $A$ is said to be *left-invertible* if a left inverse exists. Note that if $A$ has size $m \times n$, a left inverse $X$ will have size $n \times m$, the same dimensions as $A^T$.

**Examples.**

- If $A$ is a number (*i.e.*, a $1 \times 1$ matrix), then a left inverse $X$ is the same as the inverse of the number. In this case, $A$ is left-invertible whenever $A$ is nonzero, and it has only one left-inverse.
- Any nonzero $n$-vector $A$, considered as an $n \times 1$ matrix, is left invertible. For any index $i$ with $A_i \neq 0$, the row $n$-vector $X = (1/A_i)e_i^T$ satisfies $XA = 1$.
- The matrix

$$A = \begin{bmatrix} -3 & -4 \\ 4 & 6 \\ 1 & 1 \end{bmatrix}$$

has two different left inverses:

$$B = \frac{1}{9} \begin{bmatrix} -11 & -10 & 16 \\ 7 & 8 & -11 \end{bmatrix}, \qquad C = \frac{1}{2} \begin{bmatrix} 0 & -1 & 6 \\ 0 & 1 & -4 \end{bmatrix}.$$

This can be verified by checking that $BA = CA = I$. The example illustrates that a left-invertible matrix can have more than one left inverse.

- A matrix $A$ with orthonormal columns satisfies $A^T A = I$, so it is left-invertible; its transpose $A^T$ is a left inverse.

**Left-invertibility and column independence.** If $A$ has a left inverse $C$ then the columns of $A$ are linearly independent. To see this, suppose that $Ax = 0$. Multiplying on the left by a left inverse $C$, we get

$$0 = C(Ax) = (CA)x = Ix = x,$$

which shows that the only linear combination of the columns of $A$ that make 0 is the one with all coefficients zero.

We will see below that the converse is also true; a matrix has a left inverse if and only if its columns are linearly independent.

**Dimensions of left inverses.** Suppose the $m \times n$ matrix $A$ is wide, *i.e.*, $m < n$. By the independence-dimension inequality, its columns are linearly dependent, and therefore it is not left invertible. Only square or tall matrices can be left invertible.

**Solving linear equations with a left inverse.** Suppose that $Ax = b$, where $A$ is an $m \times n$ matrix and $x$ is an $n$-vector. If $C$ is a left inverse of $A$, we have

$$Cb = C(Ax) = (CA)x = Ix = x,$$

which means that $x = Cb$ is a solution of the set of linear equations. The columns of $A$ are linearly independent (since it has a left inverse), so there is only one solution of the linear equations $Ax = b$; in other words, $x = Cb$ is *the* solution of $Ax = b$.

Now suppose there is no $x$ that satisfies the linear equations $Ax = b$, and let $C$ be a left inverse of $A$. Then $x = Cb$ does not satisfy $Ax = b$, since no vector satisfies this equation by assumption. This gives a way to check if the linear equations $Ax = b$ have a solution, and to find one when there is one, provided we have a left inverse of $A$. We simply test whether $A(Cb) = b$. If this holds, then we have found a solution of the linear equations; if it does not, then we can conclude that there is no solution of $Ax = b$.

In summary, a left inverse can be used to determine whether or not a solution of an over-determined set of linear equations exists, and when it does, find the unique solution.

**Right inverse.** Now we turn to the closely related concept of right inverse. A matrix $X$ that satisfies

$$AX = I$$

is called a *right inverse* of $A$. The matrix $A$ is *right-invertible* if a right inverse exists. Any right inverse has the same dimensions as $A^T$.

**Left and right inverse of matrix transpose.** If $A$ has a right inverse $B$, then $B^T$ is a left inverse of $A^T$, since $B^T A^T = (AB)^T = I$. If $A$ has a left inverse $C$, then $C^T$ is a right inverse of $A^T$, since $A^T C^T = (CA)^T = I$. This observation allows us to map all the results for left invertibility given above to similar results for right invertibility. Some examples are given below.

- A matrix is right invertible if and only if its rows are linearly independent.
- A tall matrix cannot have a right inverse. Only square or wide matrices can be right invertible.

**Solving linear equations with a right inverse.** Consider the set of $m$ linear equations in $n$ variables $Ax = b$. Suppose $A$ is right-invertible, with right inverse $B$. This implies that $A$ is square or wide, so the linear equations $Ax = b$ are square or underdetermined.

Then for *any* $m$-vector $b$, the $n$-vector $x = Bb$ satisfies the equation $Ax = b$. To see this, we note that

$$Ax = A(Bb) = (AB)b = Ib = b.$$

We can conclude that if $A$ is right-invertible, then the linear equations $Ax = b$ can be solved for *any* vector $b$. Indeed, $x = Bb$ is a solution. (There can be other solutions of $Ax = b$; the solution $x = Bb$ is simply one of them.)

In summary, a right inverse can be used to find *a* solution of a square or underdetermined set of linear equations, for any vector $b$.

**Examples.** Consider the matrix appearing in the example above,

$$A = \begin{bmatrix} -3 & -4 \\ 4 & 6 \\ 1 & 1 \end{bmatrix}$$

and the two left inverses

$$B = \frac{1}{9}\begin{bmatrix} -11 & -10 & 16 \\ 7 & 8 & -11 \end{bmatrix}, \qquad C = \frac{1}{2}\begin{bmatrix} 0 & -1 & 6 \\ 0 & 1 & -4 \end{bmatrix}.$$

- The over-determined linear equations $Ax = (1, -2, 0)$ have the unique solution $x = (1, -1)$, which can be obtained from *either* left inverse:

  $$x = B(1, -2, 0) = C(1, -2, 0).$$

- The over-determined linear equations $Ax = (1, -1, 0)$ do not have a solution, since $x = C(1, -1, 0) = (1/2, -1/2)$ does not satisfy $Ax = (1, -1, 0)$.

- The under-determined linear equations $A^T y = (1, 2)$ has solutions

  $$B^T(1, 2) = (1/3, 2/3, 38/9), \quad C^T(1, 2) = (0, 1/2, -1).$$

  (Recall that $B^T$ and $C^T$ are both right inverses of $A^T$.) We can find a solution of $A^T y = b$ for any vector $b$.

**Left and right inverse of matrix product.** Suppose $A$ and $D$ are compatible for the matrix product $AD$ (*i.e.*, the number of columns in $A$ is equal to the number of rows in $D$.) If $A$ has a right inverse $B$ and $D$ has a right inverse $E$, then $EB$ is a right inverse of $AD$. This follows from

$$(AD)(EB) = A(DE)B = AB = I.$$

If $A$ has a left inverse $C$ and $D$ has a left inverse $F$, then $FC$ is a left inverse of $AD$. This follows from

$$(FC)(AD) = F(CA)D = FD = I.$$

## 7.2 Inverse

If a matrix is left- *and* right-invertible, then the left and right inverses are unique and equal. To see this, suppose that $AX = I$ and $YA = I$. Then we have

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

In this case we call the matrix $X = Y$ simply the *inverse* of $A$, denoted $A^{-1}$, and say that $A$ is *invertible* or *nonsingular*. A square matrix that is not invertible is called *singular*.

**Dimensions of invertible matrices.** Invertible matrices must be square. (Tall matrices are not right invertible, while wide matrices are not left invertible.) A matrix $A$ and its inverse (if it exists) satisfy

$$AA^{-1} = A^{-1}A = I.$$

If $A$ has inverse $A^{-1}$, then the inverse of $A^{-1}$ is $A$; in other words, we have $(A^{-1})^{-1} = A$. For this reason we say that $A$ and $A^{-1}$ are inverses (of each other).

**Solving linear equations with the inverse.** Consider the square system of $n$ linear equations with $n$ variables, $Ax = b$. If $A$ is invertible, then for any $n$-vector $b$,

$$x = A^{-1}b \tag{7.1}$$

is a solution of the equations. (This follows since $A^{-1}$ is a right inverse of $A$.) Moreover, it is the *only* solution of $Ax = b$. (This follows since $A^{-1}$ is a left inverse of $A$.)

In summary, the square system of linear equations $Ax = b$, with $A$ invertible, has the unique solution $x = A^{-1}b$, for any $n$-vector $b$.

One immediate conclusion we can draw from the formula (7.1) is that the solution of a square set of linear equations is a linear function of the right-hand side vector $b$.

**Invertibility conditions.** For square matrices, left-invertibility, right-invertibility, and invertibility are equivalent: If a matrix is square and left-invertible, then it is also right-invertible (and therefore invertible) and vice-versa.

To see this, suppose $A$ is an $n \times n$ matrix and left-invertible. This implies that the $n$ columns of $A$ are linearly independent. Therefore they form a basis and so any $n$-vector can be expressed as a linear combination of the columns of $A$. In particular, each of the $n$ unit vectors $e_i$ can be expressed as $e_i = Ab_i$ for some $n$-vector $b_i$. The matrix $B = \left[\begin{array}{cccc} b_1 & b_2 & \cdots & b_n \end{array}\right]$ satisfies

$$AB = \left[\begin{array}{cccc} Ab_1 & Ab_2 & \cdots & Ab_n \end{array}\right] = \left[\begin{array}{cccc} e_1 & e_2 & \cdots & e_n \end{array}\right] = I.$$

So $B$ is a right inverse of $A$.

We have just shown that for a square matrix $A$,

$$\text{left invertibility} \quad \Longrightarrow \quad \text{column independence} \quad \Longrightarrow \quad \text{right invertibility}.$$

Applying the same result to the transpose of $A$ allows us to also conclude that

$$\text{right invertibility} \quad \Longrightarrow \quad \text{row independence} \quad \Longrightarrow \quad \text{left invertibility}.$$

So all six of these conditions are equivalent; if any one of them holds, so do the other five.

In summary, for a square matrix $A$, the following are equivalent.

- $A$ is invertible.
- The columns of $A$ are linearly independent.
- The rows of $A$ are linearly independent.
- $A$ has a left inverse.
- $A$ has a right inverse.

**Examples.**

- The identity matrix $I$ is invertible, with inverse $I^{-1} = I$, since $II = I$.
- A diagonal matrix $A$ is invertible if and only if its diagonal entries are nonzero. The inverse of an $n \times n$ diagonal matrix $A$ with nonzero diagonal entries is

$$A^{-1} = \left[\begin{array}{cccc} 1/A_{11} & 0 & \cdots & 0 \\ 0 & 1/A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/A_{nn} \end{array}\right],$$

since

$$AA^{-1} = \left[\begin{array}{cccc} A_{11}/A_{11} & 0 & \cdots & 0 \\ 0 & A_{22}/A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn}/A_{nn} \end{array}\right] = I.$$

In compact notation, we have

$$\mathbf{diag}(A_{11}, \ldots, A_{nn})^{-1} = \mathbf{diag}(A_{11}^{-1}, \ldots, A_{nn}^{-1}).$$

- As a non-obvious example, the matrix

$$A = \begin{bmatrix} 1 & -2 & 3 \\ 0 & 2 & 2 \\ -3 & -4 & -4 \end{bmatrix}$$

is invertible, with inverse

$$A^{-1} = \frac{1}{30} \begin{bmatrix} 0 & -20 & -10 \\ -6 & 5 & -2 \\ 6 & 10 & 2 \end{bmatrix}.$$

This can be verified by checking that $AA^{-1} = I$ (or that $A^{-1}A = I$, since either of these implies the other).

- $2 \times 2$ *matrices.* A $2 \times 2$ matrix $A$ is invertible if and only if $A_{11}A_{22} \neq A_{12}A_{21}$, with inverse

$$A^{-1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \frac{1}{A_{11}A_{22} - A_{12}A_{21}} \begin{bmatrix} A_{22} & -A_{12} \\ -A_{21} & A_{11} \end{bmatrix}.$$

(There are similar formulas for the inverse of a matrix of any size, but they grow very quickly in complexity and so are not very useful in most applications.)

- *Orthogonal matrix.* If $A$ is square with orthonormal columns, we have $A^TA = I$, so $A$ is invertible with inverse $A^{-1} = A^T$.

**Inverse of matrix transpose.** If $A$ is invertible, its transpose $A^T$ is also invertible and its inverse is $(A^{-1})^T$:

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

Since the order of the transpose and inverse operations does not matter, this matrix is sometimes written as $A^{-T}$.

**Inverse of matrix product.** If $A$ and $B$ are invertible (hence, square) and of the same size, then $AB$ is invertible, and

$$(AB)^{-1} = B^{-1}A^{-1}. \qquad (7.2)$$

The inverse of a product is the product of the inverses, in reverse order.

**Dual basis.** Suppose that $A$ is invertible with inverse $B = A^{-1}$. Let $a_1, \ldots, a_n$ be the columns of $A$, and $b_1, \ldots, b_n$ denote the (transposes of the) *rows* of $B$, *i.e.*, the columns of $B^T$:

$$A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}, \qquad B = \begin{bmatrix} b_1^T \\ \vdots \\ b_n^T \end{bmatrix}.$$

We know that $a_1, \ldots, a_n$ form a basis, since the columns of $A$ are independent. The vectors $b_1, \ldots, b_n$ also form a basis, since the rows of $B$ are independent. They are called the *dual basis* of $a_1, \ldots, a_n$. (The dual basis of $b_1, \ldots, b_n$ is $a_1, \ldots, a_n$, so they called *dual bases*.)

The matrix equation $BA = I$ can be expressed as

$$b_i^T a_j = \left\{ \begin{array}{ll} 1 & i = j \\ 0 & i \neq j, \end{array} \right.$$

for $i, j = 1, \ldots, n$. In words, $b_i$ is orthogonal to $a_j$, for $j \neq i$, and has inner product 1 for $j = i$.

Now suppose that $x$ is an $n$-vector. It can be expressed as a linear combination of the basis vectors $a_1, \ldots, a_n$:

$$x = \beta_1 a_1 + \cdots + \beta_n a_n.$$

Multiplying this equation on the left by $b_i^T$ we get

$$b_i^T x = \beta_1 (b_i^T a_1) + \cdots + \beta_n (b_i^T a_n) = \beta_i, \quad i = 1, \ldots, n.$$

So the dual basis gives us a simple way to find the coefficients in the expansion of a vector in the $a_1, \ldots, a_n$ basis.

**Negative matrix powers.** We can now give a meaning to matrix powers with negative integer exponents. Suppose $A$ is a square invertible matrix and $k$ is a positive integer. Then by repeatedly applying property (7.2), we get

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

We denote this matrix as $A^{-k}$. For example, if $A$ is square and invertible, then $A^{-2} = A^{-1}A^{-1} = (AA)^{-2}$. With $A^0$ defined as $A^0 = I$, the identity $A^{k+l} = A^k A^l$ holds for all integers $k$ and $l$.

**Triangular matrix.** A triangular matrix with nonzero diagonal elements is invertible. We first discuss this for a lower triangular matrix. Let $L$ be $n \times n$ and lower triangular with nonzero diagonal elements. We show that the columns are linearly independent, *i.e.*, $Lx = 0$ is only possible if $x = 0$. Expanding the matrix-vector product, we can write $Lx = 0$ as

$$\begin{array}{rcl} L_{11}x_1 & = & 0 \\ L_{21}x_1 + L_{22}x_2 & = & 0 \\ L_{31}x_1 + L_{32}x_2 + L_{33}x_3 & = & 0 \\ & \vdots & \\ L_{n1}x_1 + L_{n2}x_2 + \cdots + L_{n,n-1}x_{n-1} + L_{nn}x_n & = & 0. \end{array}$$

Since $L_{11} \neq 0$, the first equation implies $x_1 = 0$. Using $x_1 = 0$, the second equation reduces to $L_{22}x_2 = 0$. Since $L_{22} \neq 0$, we conclude that $x_2 = 0$. Using $x_1 = x_2 = 0$, the third equation now reduces to $L_{33}x_3 = 0$, and since $L_{33}$ is assumed to be nonzero, we have $x_3 = 0$. Continuing this argument, we find that all entries of $x$ are zero, and this shows that the columns of $L$ are linearly independent. It follows that $L$ is invertible.

A similar argument can be followed to show that an upper triangular matrix with nonzero diagonal elements is invertible. One can also simply note that if $R$ is upper triangular, then $L = R^T$ is lower triangular with the same diagonal, and use the formula $(L^T)^{-1} = (L^{-1})^T$ for the inverse of the transpose.

**Inverse via QR factorization.** The QR factorization gives a simple expression for the inverse of an invertible matrix. If $A$ is square and invertible, its columns are linearly independent, so it has a QR factorization $A = QR$. The matrix $Q$ is orthogonal and $R$ is upper triangular with positive diagonal entries. Hence $Q$ and $R$ are invertible, and the formula for the inverse product gives

$$A^{-1} = (QR)^{-1} = R^{-1}Q^{-1} = R^{-1}Q^T. \tag{7.3}$$

In the following section we give an algorithm for computing $R^{-1}$, or more directly, the product $R^{-1}Q^T$. This gives us a method to compute the matrix inverse.

## 7.3 Solving linear equations

**Back substitution.** We start with an algorithm for solving a set of linear equations, $Rx = b$, where the $n \times n$ matrix $R$ is upper triangular with nonzero diagonal entries (hence, invertible). We write out the equations as

$$\begin{array}{rcl} R_{11}x_1 + R_{12}x_2 + \cdots + R_{1,n-1}x_{n-1} + R_{1n}x_n & = & b_1 \\ & \vdots & \\ R_{n-2,n-2}x_{n-2} + R_{n-2,n-1}x_{n-1} + R_{n-2,n}x_n & = & b_{n-2} \\ R_{n-1,n-1}x_{n-1} + R_{n-1,n}x_n & = & b_{n-1} \\ R_{nn}x_n & = & b_n. \end{array}$$

From the last equation, we find that $x_n = b_n/R_{nn}$. Now that we know $x_n$, we substitute it into the second to last equation, which gives us

$$x_{n-1} = (b_{n-1} - R_{n-1,n}x_n))/R_{n-1,n-1}.$$

We can continue this way to find $x_{n-2}, x_{n-3}, \ldots, x_1$. This algorithm is known as *back substitution*, since the variables are found one at a time, starting from $x_n$, and we substitute the ones that are known into the remaining equations.

**Algorithm 7.1** BACK SUBSTITUTION

**given** an $n \times n$ upper triangular matrix $R$ with nonzero diagonal entries, and an $n$-vector $b$.

```
For $i = n, \dots, 1$,
    $x_i = (b_i - R_{i,i+1}x_{i+1} - \cdots - R_{i,n}x_n)/R_{ii}$.
```

The back substitution algorithm computes the solution of $Rx = b$, *i.e.*, $x = R^{-1}b$. If cannot fail since the divisions in each step are by the diagonal entries of $R$, which are assumed to be nonzero.

Lower triangular matrices with nonzero diagonal elements are also invertible; we can solve equations with lower triangular invertible matrices using *forward substitution*, the obvious analog of the algorithm given above. In forward substitution, we find $x_1$ first, then $x_2$, and so on.

**Complexity of back substitution.** The first step requires 1 flop (division by $R_{nn}$). The next step requires one multiply, one subtraction, and one division, for a total of 3 flops. The $k$th step requires $k-1$ multiplies, $k-1$ subtractions, and one division, for a total of $2k-1$ flops. The total number of flops for back substitution is then

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

flops.

**Solving linear equations using the QR factorization.** The formula (7.3) for the inverse of a matrix in terms of its QR factorization suggests a method for solving a square system of linear equations $Ax = b$ with $A$ invertible. The solution

$$x = A^{-1}b = R^{-1}Q^Tb \qquad (7.4)$$

can be found by first computing the matrix vector product $y = Q^Tb$, and then solving the triangular equation $Rx = y$ by back substitution.

**Algorithm 7.2** SOLVING LINEAR EQUATIONS VIA QR FACTORIZATION

**given** an $n \times n$ invertible matrix $A$ and an $n$-vector $b$.

1. *QR factorization.* Compute the QR factorization $A = QR$.
2. Compute $Q^Tb$.
3. *Back substitution.* Solve the triangular equation $Rx = Q^Tb$ using back substitution.

The first step requires $2n^3$ flops (see §3.4), the second step requires $2n^2$ flops, and the third step requires $n^2$ flops. The total number of flops is then

$$2n^3 + 3n^2 \approx 2n^3,$$

so the order is $n^3$, cubic in the number of variables, which is the same as the number of equations.

In the complexity analysis above, we found that the first step, the QR factorization, dominates the other two; that is, the cost of the other two is negligible in comparison to the cost of the first step. This has some interesting practical implications, which we discuss below.

**Factor-solve methods.** Algorithm 7.2 is similar to many methods for solving a set of linear equations and is sometimes referred to as a *factor-solve* scheme. A factor-solve scheme consists of two steps. In the first (factor) step the data matrix is factored as a product of matrices with special properties. In the second (solve) step one or more linear equations that involve the factors in the factorization are solved. (In algorithm 7.2, the solve step consists of steps 1 and 2.) The complexity of the solve step is smaller than the complexity of the factor step, and in many cases, it is negligible by comparison. This is the case in algorithm 7.2, where the factor step has order $n^3$ and the solve step has order $n^2$.

Now suppose that we must solve several sets of linear equations,

$$Ax_1 = b_1, \ldots, Ax_k = b_k,$$

all with the same coefficient matrix $A$, but different right-hand sides. Solving the $l$ problems independently, by applying algorithm 7.2 $l$ times, costs $2ln^3$ flops. A more efficient method exploits the fact that $A$ is the same matrix in each problem, so we can re-use the matrix factorization in step 1 and only need to repeat the second step to compute $\hat{x}_k = R^{-1}Q^Tb_k$ for $k = 1, \ldots, l$. (This is sometimes referred to as *factorization caching*, since we save the factorization after carrying it out, for later use.) The cost of this method is $2n^3 + 3ln^2$ flops, or approximately $2n^3$ flops if $l \ll n$. The (surprising) conclusion is that we can solve *multiple* sets of linear equations, with the same coefficient matrix $A$, at essentially the same cost as solvng *one* set of linear equations.

**Computing the matrix inverse.** We can now describe a method to compute the inverse $B = A^{-1}$ of an (invertible) $n \times n$ matrix $A$. We first compute the QR factorization of $A$, so $A^{-1} = R^{-1}Q^T$. We can write this as $RB = Q^T$, which, written out by columns is

$$Rb_i = \tilde{q}_i, \quad i = 1, \ldots, n,$$

where $b_i$ is the $i$th column of $B$ and $\tilde{q}_i$ is the $i$th column of $Q^T$. We can solve these equations using back substitution, to get the columns of the inverse $B$.

**Algorithm 7.3** COMPUTING THE INVERSE VIA QR FACTORIZATION

```
given an $n \times n$ invertible matrix $A$.

1. QR factorization. Compute the QR factorization $A = QR$.
2. For $i = 1, \ldots, n$,
       Solve the triangular equation $Rb_i = \tilde{q}_i$ using back substitution.
```

The complexity of this method is $2n^3$ flops (for the QR factorization) and $n^3$ for $n$ back substitutions, each of which costs $n^2$ flops. So we can compute the matrix inverse in around $3n^3$ flops.

This gives an alternative method for solving the square set of linear equations $Ax = b$: We first compute the inverse matrix $A^{-1}$, and then the matrix-vector product $x = (A^{-1})b$. This method has a higher flop count than directly solving the equations using algorithm 7.2 ($3n^3$ versus $2n^3$), so algorithm 7.2 is the usual method of choice. While the matrix inverse appears in many formulas (such as the solution of a set of linear equations), it is *computed* far less often.

## 7.4 Examples

**Polynomial interpolation.** The 4-vector $c$ gives the coefficients of a cubic polynomial,

$$p(x) = c_1x^3 + c_2x^2 + c_3x + c_4$$

(see pages 80 and 63). We seek the coefficients that satisfy

$$p(-1.1) = b_1, \qquad p(-0.4) = b_2, \qquad p(0.1) = b_3, \qquad p(0.8) = b_4.$$

We can express this as the system of 4 equations in 4 variables $Ac = b$, where

$$A = \left[\begin{array}{cccc} (-1.1)^3 & (-1.1)^2 & -1.1 & 1 \\ (-0.4)^3 & (-0.4)^2 & -0.4 & 1 \\ (0.1)^3 & (0.1)^2 & 0.1 & 1 \\ (0.8)^3 & (0.8)^2 & 0.8 & 1 \end{array}\right].$$

The unique solution is $c = A^{-1}b$, where

$$A^{-1} = \left[\begin{array}{rrrr} -0.6266 & 2.3810 & -2.3810 & 0.6266 \\ 0.3133 & 0.4762 & -1.6667 & 0.8772 \\ 0.1754 & -2.1667 & 1.8095 & 0.1817 \\ -0.0201 & 0.2095 & 0.8381 & -0.0276 \end{array}\right]$$

(truncated to 4 decimal places). The columns of $A^{-1}$ are interesting: They give the coefficients of a polynomial that evaluates to 0 at three of the points, and 1 at the other point. (These are called the *Lagrange polynomials* associated with the points $-1.1, -0.4, 0.1, 0.8$.) The rows are also interesting: The $i$th row shows how the values $b_1, \ldots, b_4$ map into the coefficient $c_i$. For example, we see that the coefficient $c_4$ is not very sensitive to the value of $b_1$ (since $(A^{-1})_{41}$ is small). We can also see that for each increase of one in $b_3$, the coefficient $c_1$ decreases by around 2.4.

This is illustrated in figure 7.1, which shows the two cubic polynomials that interpolate the two sets of points shown as filled circles and squares, respectively.

**Balancing chemical equations.** (See page 80 for background.) We consider the problem of balancing the chemical reaction

$$a_1\mathrm{Cr_2O_2^{2-}} + a_2\mathrm{Fe^{2+}} + a_3\mathrm{H^+} \longrightarrow b_1\mathrm{Cr^{3+}} + b_2\mathrm{Fe^{3+}} + b_3\mathrm{H_2O},$$

where the superscript gives the charge of each reactant and product. There are 4 atoms (Cr, O, Fe, H) and charge to balance. The reactant and product matrices are (using the order just listed)

$$R = \left[\begin{array}{rrr} 2 & 0 & 0 \\ 7 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & 2 & 1 \end{array}\right], \qquad P = \left[\begin{array}{rrr} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 3 & 3 & 0 \end{array}\right].$$

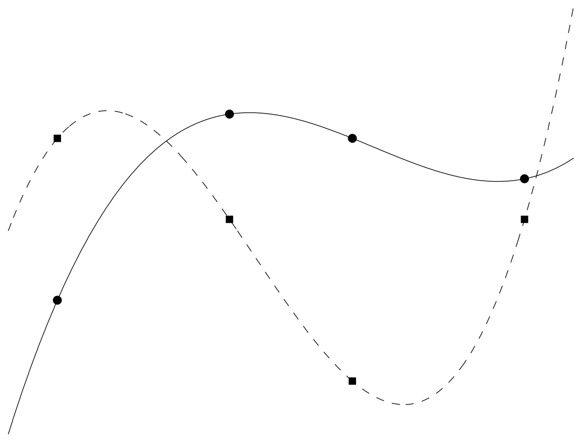

**Figure 7.1** Cubic interpolants through two sets of points, shown as circles and squares.

Imposing the condition that $a_1 = 1$ we obtain a square set of 6 linear equations,

$$\begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -2 \\ -2 & 2 & 1 & -3 & -3 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Solving these equations we obtain

$$a_1 = 1, \quad a_2 = 6, \quad a_3 = 14, \qquad b_1 = 2, \quad 2_2 = 6, \quad b_3 = 7.$$

(Setting $a_1 = 1$ could have yielded fractional values for the other coefficients, but in this case, it did not.) The balanced reaction is

$$\mathrm{Cr_2O_2^{2-}} + 6\mathrm{Fe^{2+}} + 14\mathrm{H^+} \longrightarrow 2\mathrm{Cr^{3+}} + 6\mathrm{Fe^{3+}} + 7\mathrm{H_2O}.$$

**Heat diffusion.** We consider a diffusion system as described on page 81. Some of the nodes have fixed potential, *i.e.*, $e_i$ is given; for the other nodes, the associated external source $s_i$ is zero. This would model a thermal system in which some nodes are in contact with the outside world or a heat source, which maintains their temperatures (via external heat flows) at constant values; the other nodes are internal, and have no heat sources. This gives us a set of $n$ additional equations:

$$e_i = e_i^{\mathrm{fix}}, \quad i \in \mathcal{P}, \qquad s_i = 0, \quad i \notin \mathcal{P},$$

where $\mathcal{P}$ is the set of indices of nodes with fixed potential. We can write these $n$ equations in matrix-vector form as

$$Bs + Ce = d,$$

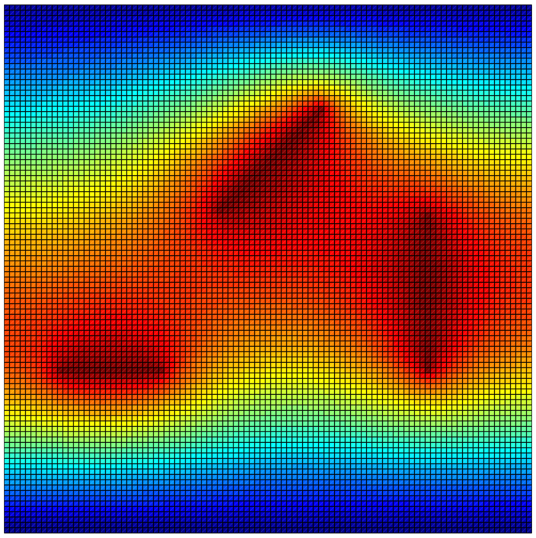

**Figure 7.2** Temperature distribution on a 100 × 100 grid of nodes. Nodes in the top and bottom rows are held at zero temperature. The three sets of nodes with rectilinear shapes are held at temperature one.

where $B$ and $C$ are the $n \times n$ diagonal matrices, and $d$ is the $n$-vector given by

$$B_{ii} = \begin{cases} 0 & i \in \mathcal{P} \\ 1 & i \notin \mathcal{P}, \end{cases} \qquad C_{ii} = \begin{cases} 1 & i \in \mathcal{P} \\ 0 & i \notin \mathcal{P}, \end{cases} \qquad d_i = \begin{cases} e_i^{\text{fix}} & i \in \mathcal{P} \\ 0 & i \notin \mathcal{P}. \end{cases}$$

We assemble the flow conservation, edge flow, and the coundary conditions into one set of $m + 2n$ equations in $m + 2n$ variables $(f, s, e)$:

$$\begin{bmatrix} A & I & 0 \\ R & 0 & A^T \\ 0 & B & C \end{bmatrix} \begin{bmatrix} f \\ s \\ e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}.$$

Assuming the coefficient matrix is invertible, we have

$$\begin{bmatrix} f \\ s \\ e \end{bmatrix} = \begin{bmatrix} A & I & 0 \\ R & 0 & A^T \\ 0 & B & C \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}.$$

This is illustrated in figure 7.2. The graph is a 100 × 100 grid, with 10000 nodes, and edges connecting each node to its horizontal and vertical neighbors. The resistance on each edge is the same. The nodes at the top and bottom are held at zero temperature, and the three sets of nodes with rectilinear shapes are held at temperature one. All other nodes have zero source value.

## 7.5 Pseudo-inverse

**Independent columns and Gram invertibility.** We first show that an $m \times n$ matrix $A$ has independent columns if and only if its $n \times n$ Gram matrix $A^T A$ is invertible.

First suppose that the columns of $A$ are independent. Let $x$ be an $n$-vector which satisfies $(A^T A)x = 0$. Multiplying on the left by $x^T$ we get

$$0 = x^T 0 = x^T (A^T Ax) = x^T A^T Ax = \|Ax\|^2,$$

which implies that $Ax = 0$. Since the columns of $A$ are independent, we conclude that $x = 0$. Since the only solution of $(A^T A)x = 0$ is $x = 0$, we conclude that $A^T A$ is invertible.

Now let's show the converse. Suppose the columns of $A$ are dependent, which means there is a nonzero $n$-vector $x$ which satisfies $Ax = 0$. Multiply on the left by $A^T$ to get $(A^T A)x = 0$. This shows that the Gram matrix $A^T A$ is singular.

**Pseudo-inverse of square or tall matrix.** We show here that if $A$ has linearly independent columns (and therefore, is square or tall) then it has a left inverse. (We already have observed the converse, that a matrix with a left inverse has independent columns.) Assuming $A$ has independent columns, we know that $A^T A$ is invertible. We now observe that the matrix $(A^T A)^{-1} A^T$ is a left inverse of $A$:

$$\left((A^T A)^{-1} A^T\right) A = (A^T A)^{-1} (A^T A) = I.$$

This particular left-inverse of $A$ will come up in the sequel, and has a name, the *pseudo-inverse* of $A$ (also called the *Moore-Penrose inverse*). It is denoted $A^\dagger$ (or $A^+$):

$$A^\dagger = (A^T A)^{-1} A^T. \qquad (7.5)$$

When $A$ is square, the pseudo-inverse $A^\dagger$ reduces to the ordinary inverse:

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

Note that this equation does not make sense (and certainly is not correct) when $A$ is not square.

**Pseudo-inverse of a square or wide matrix.** Transposing all the equations, we can show that a (square or wide) matrix $A$ has a right inverse if and only its rows are linearly independent. Indeed, one right inverse is given by

$$A^T (AA^T)^{-1}. \qquad (7.6)$$

(The matrix $AA^T$ is invertible if and only if the rows of $A$ are linearly independent.)

The matrix in (7.6) is also referred to as the pseudo-inverse of $A$, and denoted $A^\dagger$. The only possible confusion in defining the pseudo-inverse using the two different formulas (7.5) and (7.6) occurs when the matrix $A$ is square. In this case, however, they both reduce to the ordinary inverse:

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

**Pseudo-inverse in other cases.** The pseudo-inverse $A^\dagger$ is defined for any nonzero matrix, including the case when $A$ is tall but its columns are linearly dependent, the case when $A$ is wide but its rows are linearly dependent, and the case when $A$ is square but not invertible. In these cases, however, it is not a left inverse, right inverse, or inverse, respectively. Exactly what $A^\dagger$ means in these cases is beyond the scope of this book; we mention it since the reader may encounter it.

**Pseudo-inverse via QR factorization.** The QR factorization gives a simple formula for the pseudo-inverse. If $A$ is left-invertible, its columns are linearly independent and the QR factorization $A = QR$ exists. We have

$$A^TA = (QR)^T(QR) = R^TQ^TQR = R^TR,$$

so

$$A^\dagger = (A^TA)^{-1}A^T = (R^TR)^{-1}(QR)^T = R^{-1}R^{-T}R^TQ^T = R^{-1}Q^T.$$

We can compute the pseudo-inverse using the QR factorization, followed by back substitution on the columns of $Q^T$. (This is exactly the same as algorithm 7.3 when $A$ is square and invertible.) The complexity of this method is $2n^2m$ flops (for the QR factorization), and $mn^2$ flops for the $m$ back substitutions. So the total is $3mn^2$ flops.

Similarly, if $A$ is right-invertible, the QR factorization $A^T = QR$ of its transpose exists. We have $AA^T = (QR)^T(QR) = R^TQ^TQR = R^TR$ and

$$A^\dagger = A^T(AA^T)^{-1} = QR(R^TR)^{-1} = QRR^{-1}R^{-T} = QR^{-T}.$$

We can compute it using the method described above, using the formula

$$(A^T)^\dagger = (A^\dagger)^T.$$

**Solving over- and under-determined systems of linear equations.** The pseudo-inverse gives us a method for solving over-determined and under-determined systems of linear equations, provided the columns of the coefficient matrix are independent (in the over-determined case), or the rows are independent (in the under-determined case). If the columns of $A$ are independent, and over-determined equations $Ax = b$ have a solution, then $x = A^\dagger b$ is it. If the rows of $A$ are independent, the under-determined equations $Ax = b$ have a solution for any vector $b$, and $x = A^\dagger b$ is a solution.

# Chapter 8

# Least-squares

In this chapter we look at the powerful idea, due to the famous mathematician Gauss, of finding approximate solutions of over-determined systems of linear equations by minimizing the sum of the squares of the errors in the equations. The method, and some extensions we describe, are widely used in many applications areas.

## 8.1 Linear least-squares problem

Suppose that the $m \times n$ matrix $A$ is tall, so the system of linear equations $Ax = b$, where $b$ is an $m$-vector, is over-determined: There are more equations ($m$) than variables to choose ($n$). These equations have a solution only if $b$ is a linear combination of the columns of $A$.

For most choices of $b$, however, there is no $n$-vector $x$ for which $Ax = b$. As a compromise, we seek an $x$ for which $r = Ax - b$, which we call the *residual* (for the equations $Ax = b$), is as small as possible. This suggests that we should choose $x$ so as to minimize the norm of the residual $\|Ax - b\|$. If we find an $x$ for which the residual vector is small, we have $Ax \approx b$, *i.e.*, $x$ almost satisfies the linear equations $Ax = b$.

Minimizing the norm of the residual and its square are the same, so we can just as well minimize

$$\|Ax - b\|^2 = \|r\|^2 = r_1^2 + \cdots + r_m^2,$$

the sum of squares of the residuals. The problem of finding an $n$-vector $\hat{x}$ that minimizes $\|Ax - b\|^2$, over all possible choices of $x$, is called the *least-squares problem*. It is denoted with the notation

$$\text{minimize} \quad \|Ax - b\|^2, \tag{8.1}$$

where we should specify that the *variable* is $x$ (meaning that we should choose $x$). The matrix $A$ and the vector $b$ are called the *data* for the problem (8.1), which means that they are given to us when we are asked to find $x$. The quantity to be minimized, $\|Ax-b\|^2$, is called the *objective function* (or just objective) of the least-squares problem (8.1).

The problem (8.1) is sometimes called *linear* least-squares to emphasize that the residual $r$ (whose norm squared we are to minimize) is affine, and to distinguish it from *nonlinear* least-squares problems, in which we allow the residual $r$ to be an arbitrary function of $x$.

Any vector $\hat{x}$ that satisfies $\|A\hat{x}-b\|^2 \leq \|Ax-b\|^2$ for all $x$ is a *solution* of the least-squares problem (8.1). Such a vector is called a *least-squares approximate solution* of $Ax=b$. It is very important to understand that a least-squares approximate solution $\hat{x}$ of $Ax=b$ need not satisfy the equations $A\hat{x}=b$; it simply makes the norm of the residual as small as it can be. Some authors use the confusing phrase '$\hat{x}$ solves $Ax=b$ in the least-squares sense', but we emphasize that a least-squares approximate solution $\hat{x}$ does not, in general, solve the equation $Ax=b$.

If $\|A\hat{x}-b\|$ (which we call the *optimal residual norm*) is small, then we can say that $\hat{x}$ *approximately* solves $Ax=b$. On the other hand, if there is an $n$-vector $x$ that satisfies $Ax=b$, then it is a solution of the least-squares problem, since its associated residual norm is zero.

Another name for the least-squares problem (8.1), typically used in data fitting applications, is *regression*. We say that $\hat{x}$, a solution of the least-squares problem, is the result of *regressing* the vector $b$ onto the columns of $A$.

**Column interpretation.** If the columns of $A$ are the $m$-vectors $a_1, \ldots, a_n$, then the least-squares problem (8.1) is the problem of finding a linear combination of the columns that is closest to the $m$-vector $b$; the vector $x$ gives the coefficients:

$$\|Ax-b\|^2 = \|(x_1a_1+\cdots+x_na_n)-b\|^2.$$

If $\hat{x}$ is a solution of the least-squares problem, then the vector

$$A\hat{x} = \hat{x}_1a_1+\cdots+\hat{x}_na_n$$

is closest to the vector $b$, among all linear combinations of the vectors $a_1, \ldots, a_n$.

**Row interpretation.** Suppose the rows of $A$ are the $n$-vectors $\tilde{a}_1^T, \ldots, \tilde{a}_m^T$, so the residual components are given by

$$r_i = \tilde{a}_i^Tx - b_i, \quad i=1,\ldots,m.$$

The least-squares objective is then

$$\|Ax-b\|^2 = (\tilde{a}_1^Tx-b_1)^2+\cdots+(\tilde{a}_m^Tx-b_m)^2,$$

the sum of the squares of the residuals in $m$ scalar linear equations. Minimizing this sum of squares of the residuals is a reasonable compromise if our goal is to choose $x$ so that all of them are small.

## 8.2 Solution of least-squares problem

In this section we derive several expressions for the solution of the least-squares problem (8.1), under one assumption on the data matrix $A$:

$$\textit{The columns of } A \textit{ are linearly independent.} \tag{8.2}$$

**Solution via calculus.** In this section we find the solution of the least-squares problem using some basic results from calculus. (We will also give an independent verification of the result, that does not rely on calculus, below.) From calculus we know that any minimizer $\hat{x}$ of the function $f(x) = \|Ax - b\|^2$ must satisfy

$$\frac{\partial f}{\partial x_i}(\hat{x}) = 0, \quad i = 1, \ldots, n,$$

which we can express as the vector equation

$$\nabla f(\hat{x}) = 0,$$

where $\nabla f(\hat{x})$ is the gradient of $f$ evaluated at $\hat{x}$. This gradient can be expressed in matrix form as

$$\nabla f(x) = 2A^T(Ax - b). \tag{8.3}$$

For completeness we derive the formula (8.3). Writing the least-squares objective out as a sum, we get

$$f(x) = \|Ax - b\|^2 = \sum_{i=1}^m \left( \sum_{j=1}^n A_{ij}x_j - b_i \right)^2.$$

To find $\nabla f(x)_k$ we take the partial derivative of $f$ with respect to $x_k$. Differentiating the sum term by term, we get

$$\begin{aligned} \nabla f(x)_k &= \frac{\partial f}{\partial x_k}(x) \\ &= \sum_{i=1}^m 2 \left( \sum_{j=1}^n A_{ij}x_j - b_i \right) (A_{ik}) \\ &= \sum_{i=1}^m 2(A^T)_{ki}(Ax - b)_i \\ &= \left(2A^T(Ax - b)\right)_k. \end{aligned}$$

This is our formula (8.3), written out in terms of its components.

Now we continue the derivation of the solution of the least-squares problem. Any minimizer $\hat{x}$ of $\|Ax - b\|^2$ must satisfy

$$\nabla f(\hat{x}) = 2A^T(A\hat{x} - b) = 0,$$

which can be written as

$$A^TA\hat{x} = A^Tb. \tag{8.4}$$

These equations are called the *normal equations*. The coefficient matrix $A^T A$ is the Gram matrix associated with the columns of $A$; its entries are inner products of columns of $A$.

Our assumption (8.2) that the columns of $A$ are independent implies that the Gram matrix $A^T A$ is invertible (§7.5, page 110). This implies that

$$\hat{x} = (A^T A)^{-1} A^T b \tag{8.5}$$

is the only solution of the normal equations (8.4). So this must be the unique solution of the least-squares problem (8.1). We have already encountered the matrix $(A^T A)^{-1} A^T$ that appears in this equations: It is the pseudo-inverse of the matrix $A$, given in (7.5).

So we can write the solution of the least-squares problem in the simple form

$$\hat{x} = A^\dagger b. \tag{8.6}$$

We observed in §7.5 that $A^\dagger$ is a left inverse of $A$, which means that $\hat{x} = A^\dagger b$ solves $Ax = b$ if this set of over-determined equations has a solution. But now we see that $\hat{x} = A^\dagger b$ is the least-squares approximate solution, *i.e.*, it minimizes $\|Ax - b\|^2$. (And if there is a solution of $Ax = b$, then $\hat{x} = A^\dagger b$ is it.)

The equation (8.6) looks very much like the formula for solution of the linear equations $Ax = b$, when $A$ is square and invertible, *i.e.*, $x = A^{-1} b$. It is very important to understand the difference between the formula (8.6) for the least-squares approximate solution, and the formula for the solution of a square set of linear equations, $x = A^{-1} b$. In the case of linear equations and the inverse, $x = A^{-1} b$ actually satisfies $Ax = b$. In the case of the least-squares approximate solution, $\hat{x} = A^\dagger b$ generally *does not* satisfy $A\hat{x} = b$.

The formula (8.6) shows us that the solution $\hat{x}$ of the least-squares problem is a *linear* function of $b$. This generalizes the fact that the solution of a square invertible set of linear equations is a linear function of its right-hand side.

**Direct verification of least-squares solution.** In this section we directly show that $\hat{x} = (A^T A)^{-1} b$ is the solution of the least-squares problem (8.1), without relying on calculus. We will show that for any $x \neq \hat{x}$, we have

$$\|A\hat{x} - b\|^2 < \|Ax - b\|^2,$$

establishing that $\hat{x}$ is the unique vector that minimizes $\|Ax - b\|^2$.

We start by writing

$$\begin{aligned} \|Ax - b\|^2 &= \|(Ax - A\hat{x}) + (A\hat{x} - b)\|^2 \\ &= \|Ax - A\hat{x}\|^2 + \|A\hat{x} - b\|^2 + 2(Ax - A\hat{x})^T (A\hat{x} - b), \end{aligned} \tag{8.7}$$

where we use the identity $\|u + v\|^2 = (u + v)^T (u + v) = \|u\|^2 + \|v\|^2 + 2u^T v$. The third term in (8.7) is zero:

$$\begin{aligned} (Ax - A\hat{x})^T (A\hat{x} - b) &= (x - \hat{x})^T A^T (A\hat{x} - b) \\ &= (x - \hat{x})^T (A^T A\hat{x} - A^T b) \\ &= (x - \hat{x})^T 0 \\ &= 0, \end{aligned}$$

where we use $(A^TA)\hat{x} = A^Tb$ in the third line. With this simplification, (8.7) reduces to

$$\|Ax - b\|^2 = \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2.$$

The first term on the right-hand side is nonnegative and therefore

$$\|Ax - b\|^2 \geq \|A\hat{x} - b\|^2.$$

This shows that $\hat{x}$ minimizes $\|Ax-b\|^2$; we now show that it is the unique minimizer. Suppose equality holds above, that is, $\|Ax - b\|^2 = \|A\hat{x} - b\|^2$. Then we have $\|A(x - \hat{x})\|^2 = 0$, which implies $A(x - \hat{x}) = 0$. Since $A$ has independent columns, we conclude that $x - \hat{x} = 0$, *i.e.*, $x = \hat{x}$. So the only $x$ with $\|Ax - b\|^2 = \|A\hat{x} - b\|^2$ is $x = \hat{x}$; for all $x \neq \hat{x}$, we have $\|Ax - b\|^2 > \|A\hat{x} - b\|^2$.

**Computing least-squares approximate solutions.** We can use the QR factorization to compute the least-squares approximate solution (8.5). Let $A = QR$ be the QR factorization of $A$ (which exists by our assumption (8.2) that its columns are independent). We have already seen that the pseudo-inverse $A^\dagger$ can be expressed as $A^\dagger = R^{-1}Q^T$, so we have

$$\hat{x} = R^{-1}Q^Tb. \tag{8.8}$$

To compute $\hat{x}$ we first multiply $b$ by $Q^T$; then we compute $R^{-1}(Q^Tb)$ using back substitution. This is summarized in the following algorithm, which returns the least-squares approximate solution $\hat{x}$, given $A$ and $b$.

**Algorithm 8.1** Least-squares via QR factorization

**given** an $m \times n$ matrix $A$ with linearly independent columns and an $m$-vector $b$.

1. *QR factorization.* Compute the QR factorization $A = QR$.
2. Compute $Q^Tb$.
3. *Back subsitution.* Solve the triangular equation $R\hat{x} = Q^Tb$.

The complexity of the first step is $2mn^2$ flops. The second step involves a matrix-vector multiplication, which takes $2mn$ flops. The third step requires $n^2$ flops. The total number of flops is

$$2mn^2 + 2mn + n^2 \approx 2mn^2,$$

neglecting the second and third terms, which are smaller than the first by factors of $n$ and $2m$, respectively. The order of the algorithm is $mn^2$. The complexity is linear in the row dimension of $A$ and quadratic in the number of variables.

Algorithm 8.1 is another example of a factor-solve algorithm. Suppose we need to solve several least-squares problems

$$\text{minimize} \quad \|Ax_k - b_k\|^2,$$

$k = 1, \ldots, l$, which have the same data matrix $A$ but different vectors $b_k$. If we re-use the QR factorization of $A$ the cost is $2mn^2 + l(2mn + n^2)$ flops. When $l$ is small compared to $n$ this is roughly $2mn^2$ flops, the same cost as a single least-squares problem.

**Comparison to solving a square system of linear equations.** Recall that the solution of the square invertible system of linear equations $Ax = b$ is $x = A^{-1}b$. We can express $x$ using the QR factorization of $A$ as $x = R^{-1}Q^Tb$ (see (7.4)). This equation is formally identical to (8.8). The only difference is that in (8.8), $A$ and $Q$ need not be square, and $R^{-1}Q^Tb$ is the least-squares approximate solution, which is not (in general) a solution of $Ax = b$.

Indeed, algorithm 8.1 is formally the same as algorithm 7.2, the QR factorization method for solving linear equations. (The only difference is that in algorithm 8.1, $A$ and $Q$ can be tall.)

When $A$ is square, solving the linear equations $Ax = b$ and the least-squares problem of minimizing $\|Ax-b\|^2$ are the same, and algorithm 7.2 and algorithm 8.1 are the same. So we can think of algorithm8.1 as a generalization of algorithm 7.2, which solves the equation $Ax = b$ when $A$ is square, and computes the least-squares approximate solution when $A$ is tall.

## 8.3 Least-squares data fitting

Least-squares is widely used as a method to construct a mathematical model from some data, experiments, or observations. Suppose we have an $n$-vector $x$, and a scalar $y$, and we believe that they are related, perhaps approximately, by some function $f : \mathbf{R}^n \to \mathbf{R}$:

$$y \approx f(x).$$

The vector $x$ might represent a set of $n$ feature values, and is called the *feature vector* or the vector of *independent variables*, depending on the context. The scalar $y$ represents some *outcome* (also called *response variable*) that we are interested in. Or $x$ could represent the previous $n$ values of a time series, and $y$ represents the next value.

We don't know $f$, but we might have some idea about its general form. But we do have some *data*, given by

$$(x_1, y_1), \ldots, (x_N, y_N).$$

These data are also called *observations*, *examples*, *samples*, or *measurements*, depending on the context.

We will form a *model* of the relationship between $x$ and $y$, given by

$$y \approx \hat{f}(x),$$

where $\hat{f} : \mathbf{R}^n \to \mathbf{R}$. We write $\hat{y} = \hat{f}(x)$, where $\hat{y}$ is the (scalar) *prediction* (of the outcome $y$), given the independent variable (vector) $x$.

The model has the form

$$\hat{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x),$$

where $f_i : \mathbf{R}^n \to \mathbf{R}$ are *basis functions* that we choose, and $\theta_i$ are the *model parameters* that we choose. This form of model is called *linear in the parameters*, since for each $x$, $\hat{f}(x)$ is a linear function of the model parameter $p$-vector $\theta$. The basis functions are usually chosen based on our idea of what $f$ looks like. (We will see examples of this below.) Once the basis functions have been chosen, there is the question of how to choose the model parameters, given our set of data.

Our goal is to choose the model $\hat{f}$ so that it is consistent with the data, *i.e.*, we have $y_i \approx \hat{f}(x_i)$, for $i = 1, \ldots, N$. (There is another goal in choosing $\hat{f}$, that we will discuss in §8.5.) For data sample $i$, our model predicts the value $\hat{y}_i = \hat{f}(x_i)$, so the *prediction error* or *residual* for this data point is

$$r_i = \hat{y}_i - y_i.$$

Let $r$, $y$, and $\hat{y}$ denote the $N$-vectors with entries $r_i$, $y_i$, and $\hat{y}_i$, respectively. (In the notation above, the symbols $y$ and $\hat{y}$ refer to generic scalar values; but now we use them to denote $N$-vectors of those values, for the observed data value.) With this notation we can express the (vector of) residuals as $r = \hat{y} - y$. A natural measure of how well the model predicts the observed data, or how consistent it is with the observed data, is the RMS prediction error $\mathbf{rms}(r)$. The ratio $\mathbf{rms}(r)/\,\mathbf{rms}(y)$ gives a relative prediction error. For example, if the relative prediction error is 0.1, we might say that the model predicts the outcomes, or fits the data, within 10%.

A very common method for choosing the model parameters is to minimize the RMS prediction error, which is the same as minimizing the sum of squares of the predictions errors, $\|r\|^2$. We now show that this is a least-squares problem. Expressing $\hat{f}(x_i)$ in terms of the model parameters, we have

$$\hat{y}_i = A_{i1}\theta_1 + \cdots + A_{ip}\theta_p, \quad i = 1, \ldots, N,$$

where we define the $N \times p$ matrix $A$ as

$$A_{ij} = \hat{f}_j(x_i), \quad i = 1, \ldots, N, \quad j = 1, \ldots, p,$$

and the $p$-vector $\theta$ as $\theta = (\theta_1, \ldots, \theta_p)$. Then we have, in matrix-vector notation,

$$\hat{y} = A\theta.$$

This simple equation shows how our choice of model parameters maps into the vector of predicted values of the outcomes in the $N$ differerent experiments.

The sum of squares of the residuals is then

$$\|r\|^2 = \|A\theta - y\|^2.$$

Choosing $\theta$ to minimize this is evidently a least-squares problem, of the same form as (8.1). Provided the columns of $A$ are independent, we can solve this least-squares problem to find $\hat{\theta}$, the model parameter values that minimize the norm of the prediction error on our data set, as

$$\hat{\theta} = (A^TA)^{-1}A^Ty. \tag{8.9}$$

We say that the model parameter values $\hat{\theta}$ are obtained by *least-squares fitting on the data set.* We can interpret each term in $\|A\theta - y\|^2$. The term $\hat{y} = A\theta$ is the $m$-vector of measurements or outcomes that is predicted by our model, with the parameter vector $\theta$. The term $y$ is the $N$-vector of actual observed or measured outcomes. The difference $A\theta - y$ is the $N$-vector of prediction errors. Finally, $\|A\theta - y\|^2$ is the sum of squares of the prediction errors. This is minimized by the least-squares fit $\theta = \hat{\theta}$.

The number $\|A\hat{\theta} - y\|^2$ is called the minimum sum square error (for the given model basis and data set). The number $\|A\hat{\theta} - y\|^2/N$ is called the *minimum mean square error* (MMSE) (of our model, on the data set). Its squareroot is the minimum RMS fitting error. The model performance on the data set can be visualized by plotting $\hat{y}_i$ versus $y_i$ on a scatter plot, with a dashed line showing $\hat{y} = y$ for reference.

**Least-squares fit with a constant.** We start with the simplest possible fit: We take $p = 1$, with $f_1(x) = 1$ for all $x$. In this case the model $\hat{f}$ is a constant function, with $\hat{f}(x) = \theta_1$ for all $x$. Least-squares fitting in this case is the same as choosing the best constant value $\theta_1$ to approximate the data $y_1, \ldots, y_N$.

In this simple case, $A$ is the $N \times 1$ matrix $\mathbf{1}$, which always has independent columns (since it has one column, which is nonzero). The formula (8.9) is then

$$\hat{\theta}_1 = (A^T A)^{-1} A^T y = N^{-1} \mathbf{1}^T y = \mathbf{avg}(y),$$

where we use $\mathbf{1}^T \mathbf{1} = N$. So the best constant fit to the data is simply its mean,

$$\hat{f}(x) = \mathbf{avg}(y).$$

The RMS fit to the data (*i.e.*, the RMS value of the optimal residual) is

$$\mathbf{rms}(\mathbf{avg}(y)\mathbf{1} - y) = \mathbf{std}(y),$$

the standard deviation of the data. This gives a nice interpretation of the average value and the standard deviation of the outcomes.

It is common to compare the RMS fitting error for a more sophisticated model with the standard deviation of the outcomes, which is the optimal RMS fitting error for a constant model.

**Independent column assumption.** To use least-squares fitting we assume that the columns of the matrix $A$ are independent. We can give an interesting interpretation of what it means when this assumption fails. If the columns of $A$ are dependent, it means that one of the columns can be expressed as a linear combination of the others. Suppose, for example, that the last column can be expressed as a linear combination of the first $p-1$ columns. Using $A_{ij} = \hat{f}_j(x_i)$, this means

$$\hat{f}_p(x_i) = \beta_1 \hat{f}_1(x_i) + \cdots + \beta_{p-1} \hat{f}_{p-1}(x_i), \quad i = 1, \ldots, N.$$

This says that the value of $p$th basis function can be expressed as a linear combination of the values of the first $p-1$ basis functions *on the given data set.* Evidently, then, the $p$th basis function is redundant (on the given data set).

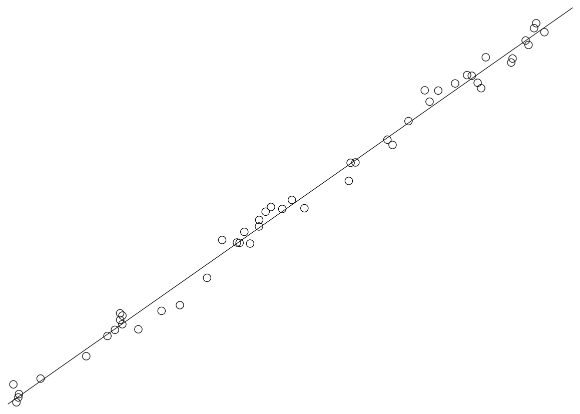

**Figure 8.1** Least-squares fit of a straight line to 50 points in a plane.

### 8.3.1 Fitting univariate functions

Suppose that $n = 1$, so both $x$ and $y$ are scalars. The relationship $y \approx f(x)$ says that $y$ is approximately a (univariate) function $f$ of $x$. We can plot the data $(x_i, y_i)$ as points in the $(x, y)$ plane, and we can plot the model $\hat{f}$ as a curve in the $(x, y)$-plane. This allows us to visualize the fit of our model to the data.

**Straight line fit.** We take basis functions $f_1(x) = 1$ and $f_2(x) = x$. Our model has the form

$$\hat{f}(x) = \theta_1 + \theta_2 x,$$

which is a straight line when plotted. (This is perhaps why $\hat{f}$ is sometimes called a linear model, even though it is in general an affine, and not linear, function of $x$.) Figure 8.1 shows an example. The matrix $A$ is given by

$$A = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}.$$

Provided that there are at least two different values appearing in $x_1, \ldots, x_N$, this matrix has independent columns. The parameters in the optimal straight-line fit to the data is given by

$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = (A^T A)^{-1} A^T y.$$

This expression is simple enough for us to work it out explicitly, although there is no computational advantage to doing so. The Gram matrix is

$$A^T A = \begin{bmatrix} N & \mathbf{1}^T x \\ \mathbf{1}^T x & x^T x \end{bmatrix},$$

where $x$ is the $N$-vector of values $(x_1, \ldots, x_N)$. The 2-vector $A^T y$ is

$$A^T y = \left[\begin{array}{c} \mathbf{1}^T y \\ x^T y \end{array}\right],$$

so we have (using the formula for the inverse of a $2 \times 2$ matrix)

$$\left[\begin{array}{c} \hat{\theta}_1 \\ \hat{\theta}_2 \end{array}\right] = \frac{1}{N x^T x - (\mathbf{1}^T x)^2} \left[\begin{array}{cc} x^T x & -\mathbf{1}^T x \\ -\mathbf{1}^T x & N \end{array}\right] \left[\begin{array}{c} \mathbf{1}^T y \\ x^T y \end{array}\right].$$

Multiplying the scalar term by $N^2$, and dividing the matrix and vector terms by $N$, we can express this as

$$\left[\begin{array}{c} \hat{\theta}_1 \\ \hat{\theta}_2 \end{array}\right] = \frac{1}{\mathbf{rms}(x)^2 - \mathbf{avg}(x)^2} \left[\begin{array}{cc} \mathbf{rms}(x)^2 & -\mathbf{avg}(x) \\ -\mathbf{avg}(x) & 1 \end{array}\right] \left[\begin{array}{c} \mathbf{avg}(y) \\ x^T y / N \end{array}\right].$$

The optimal slope $\hat{\theta}_2$ of the straight line fit can be expressed more simply in terms of the correlation coefficient $\rho$ between the data vectors $x$ and $y$, and their standard deviations. We have

$$\begin{aligned} \hat{\theta}_2 &= \frac{N x^T y - (\mathbf{1}^T x)(\mathbf{1}^T y)}{N x^T x - (\mathbf{1}^T x)^2} \\ &= \frac{(x - \mathbf{avg}(x)\mathbf{1})^T (y - \mathbf{avg}(y)\mathbf{1})}{\|x - \mathbf{avg}(x)\mathbf{1}\|^2} \\ &= \frac{\mathbf{std}(y)}{\mathbf{std}(x)} \rho. \end{aligned}$$

In the last step we used the definitions

$$\rho = \frac{(x - \mathbf{avg}(x)\mathbf{1})^T (y - \mathbf{avg}(y)\mathbf{1})}{N\, \mathbf{std}(x)\, \mathbf{std}(y)}, \qquad \mathbf{std}(x) = \frac{\|x - \mathbf{avg}(x)\mathbf{1}\|}{\sqrt{N}}$$

from chapter 2. From the first of the two normal equations, $N\theta_1 + (\mathbf{1}^T x)\theta_2 = \mathbf{1}^T y$, we also obtain a simple expression for $\hat{\theta}_1$:

$$\hat{\theta}_1 = \mathbf{avg}(y) - \hat{\theta}_2\, \mathbf{avg}(x).$$

Putting these results together, we can write the least-squares fit as

$$\hat{f}(u) = \mathbf{avg}(y) + \rho \frac{\mathbf{std}(y)}{\mathbf{std}(x)} (u - \mathbf{avg}(x)).$$

This can be expressed in the more symmetric form

$$\frac{\hat{f}(u) - \mathbf{avg}(y)}{\mathbf{std}(y)} = \rho \frac{u - \mathbf{avg}(x)}{\mathbf{std}(x)},$$

which has a nice interpretation. The left-hand side is the difference between the predicted response value and the mean response value, divided by its standard deviation. The right-hand side is the correlation coefficient $\rho$ times the same quantity, computed for the dependent variable.

The least-squares straight-line fit is used in many application areas.

**Asset $\alpha$ and $\beta$ in finance.** In finance, the straight-line fit is used to compare the returns of an individual asset, $y_i$, to the returns of the whole market, $x_i$. (The return of the whole market is typically taken to be a sum of the individual asset returns, weighted by their capitalizations.) The straight-line model for predicting the asset return $y$ from the market return $x$ is typically written in the form

$$\hat{y} = (r^{\mathrm{rf}} + \alpha) + \beta(x - \mathbf{avg}(x)),$$

where $r^{\mathrm{rf}}$ is the risk-free interest rate over the period. (Comparing this to our straight-line model, we find that $\theta_2 = \beta$, and $\theta_1 = r^{\mathrm{rf}} + \alpha - \beta\,\mathbf{avg}(x)$.) The parameter $\beta$ tells us how much of the return of the particular asset is explained, or predicted, by the market return, and the parameter $\alpha$ tells us what the average return is, over and above the risk-free interest rate. This model is so common that the terms 'Alpha' and 'Beta' are widely used in finance. (Though not always with exactly the same meaning, since there a few variations on how the parameters are defined.)

**Time series trend.** Suppose the data represents a series of samples of a quantity $y$ at time (epoch) $x_i = i$. The straight-line fit to time series data,

$$\hat{y}_i = \theta_1 + \theta_2 i, \quad i = 1, \ldots, N,$$

is called the *trend line*. Its slope, which is $\theta_2$, is interpreted as the *trend* in the quantity over time. Subtracting the trend line from the original time series we get the *de-trended time series*, $y - \hat{y}$.

**Polynomial fit.** A simple extension beyond the straight-line fit is a *polynomial fit*, with

$$f_i(x) = x^{i-1}, \quad i = 1, \ldots, p,$$

so $\hat{f}$ is a polynomial of degree at most $p - 1$,

$$\hat{f}(x) = \theta_1 + \theta_2 x + \cdots + \theta_p x^{p-1}.$$

In this case the matrix $A$ has the form

$$A = \begin{bmatrix} 1 & x_1 & \cdots & x_1^{p-1} \\ 1 & x_2 & \cdots & x_3^{p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_N & \cdots & x_N^{p-1} \end{bmatrix},$$

*i.e.*, it is a Vandermonde matrix (see (4.3). Its columns are independent provided the numbers $x_1, \ldots, x_N$ include at least $p$ different values. (See exercise **??**.) Figure 8.2 shows an example of the least-squares fit of polynomials of degree 3, 5, and 10 to a set of 20 data points. Since any polynomial of degree less than $r$ is also a polynomial of degree less than $s$, for $r \leq s$, it follows that the RMS fit attained by a polynomial with a larger degree is smaller (or at least, no larger) than that obtained by a fit with a smaller degree polynomial. This suggests that we should use the largest degree polynomial that we can, since this results in the smallest residual and the best RMS fit. But we will see in §8.5 that this is not true, and explore rational methods for choosing a model from among several candidates.

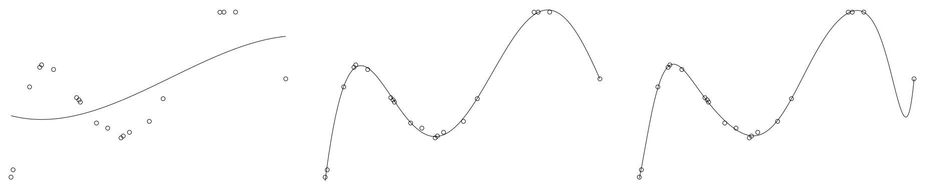

**Figure 8.2** Least-squares polynomial fits of degree 3, 5, and 10 to 20 points.

### 8.3.2 Regression

Recall that the regression model has the form

$$\hat{y} = x^T \beta + v,$$

where $\beta$ is the weight vector and $v$ is the offset. We can put this model in our general data fitting form using the basis functions $f_1(x) = 1$, and

$$f_i(x) = x_{i-1}, \quad i = 2, \ldots, n+1,$$

so $p = n+1$. The regression model can then be expressed as

$$\hat{y} = x^T \theta_{2:n} + \theta_1.$$

The matrix $A$ in our general data fitting form is given by

$$A = \left[ \begin{array}{c} X \\ \mathbf{1}^T \end{array} \right]^T = \left[ \begin{array}{cc} X^T & \mathbf{1} \end{array} \right],$$

where $X$ is the feature matrix with columns $x_1, \ldots, x_N$.

If we employ the standard trick of adding a first feature, which has the constant value one, the regression model has the simpler form $\hat{y} = x^T \beta$, where $\beta_1$ is the offset. This can be expressed in the data fitting form using $\theta = \beta$, with the matrix $A$ given by $A = X^T$, the transpose of the (augmented) feature matrix.

## 8.4 Least-squares classification

### 8.4.1 Boolean classification

In the data fitting problem, the goal is to reproduce or predict the outcome $y$, which is a (scalar) number. In a *classification problem*, the outcome $y$ takes on only a finite number of values, and for this reason is sometimes called a *label*. In the simplest case, $y$ has only two values, for example TRUE or FALSE, or SPAM or NOT SPAM. This is called the *2-way classification problem*, or the *Boolean classification problem*. We start by considering the Boolean classification problem.

We will encode $y$ as a real number, taking $y = +1$ to mean TRUE and $y = -1$ to mean FALSE. As in real-valued data fitting, we assume that an approximate relationship of the form $y \approx f(x)$ holds, where $f : \mathbf{R}^n \to \{-1 + 1\}$. (This notation means that the function $f$ takes an $n$-vector argument, and gives a resulting value that is either $+1$ or $-1$.) Our model will have the form $\hat{y} = \hat{f}(x)$, where $\hat{f} : \mathbf{R}^n \to \{-1, +1\}$. The model $\hat{f}$ is also called a *classifier*, since it classifies $n$-vectors into those for which $\hat{f}(x) = +1$ and those for which $\hat{f}(x) = -1$.

**Examples.** Boolean classifiers are widely used in many application areas.

- *Email spam detection.* The vector $x$ contains features of an email message. It can include word counts in the body of the email message, as well as other features such as the number of exclamation points and all-capital words, as well as features related to the origin of the email. The outcome is $+1$ if the message is SPAM, and $-1$ otherwise. The data used to create the classifier comes from users who have explcitly marked some messages as junk.

- *Fraud detection.* The vector $x$ gives a set of features associated with a credit card holder, such as her average monthly spending levels, median price of purchases over the last week, number of purchases in different categories, average balance, and so on, as well as some features associated with a particular proposed transaction. The outcome $y$ is $+1$ for a fraudulent transaction, and $-1$ otherwise. The data used to create the classifier is taken from historical data, that includes (some) examples of transactions that were later verified to be fraudulent and (many) that were verified to be bona fide.

- *Boolean document classification.* The vector $x$ is a word count (or histogram) vector for a document, and the outcome $y$ is $+1$ if the document has some specific topic (say, politics) and $-1$ otherwise. The data used to construct the classifier might come from a corpus of document with their topics labeled.

**Prediction errors,** For a given data point $(x, y)$, with predicted outcome $\hat{y} = \hat{f}(x)$, there are only four possibilities:

- *True positive.* $y = +1$ and $\hat{y} = +1$.
- *True negative.* $y = -1$ and $\hat{y} = -1$.
- *False positive.* $y = -1$ and $\hat{y} = +1$.
- *False negative.* $y = +1$ and $\hat{y} = -1$.

In the first two cases the predicted label is correct, and in the last two cases, the predicted label is an error. We refer to the third case as a *false positive* or *type I error*, and we refer to the fourth case as a *false negative* or *type II error.* In some applications we care equally about making the two types of errors; in others we may care more about making one type of error than another.

**Error rate and confusion matrix.** For a given data set $(x_1, y_1), \ldots, (x_N, y_N)$ and model $\hat{f}$, we can count the numbers of each of the four possibilities that occur across the data set, and display them in a *contingency table* or *confusion matrix*, which is a $2 \times 2$ table with the columns corresponding to the value of $\hat{y}_i$ and the rows corresponding to the value of $y_i$. (This is the convention used in machine learning; in statistics, the rows and columns are sometimes reversed.) The entries give the total number of each of the four cases listed above:

| | $\hat{y} = +1$ | $\hat{y} = -1$ | total |
|---|---|---|---|
| $y = +1$ | $N_{\rm tp}$ | $N_{\rm fn}$ | $N_{\rm p}$ |
| $y = -1$ | $N_{\rm fp}$ | $N_{\rm tn}$ | $N_{\rm n}$ |
| total | $N_{\rm tp} + N_{\rm fp}$ | $N_{\rm fn} + N_{\rm tp}$ | $N$ |

The diagonal entries correspond to correct decisions, with the upper left entry the number of true positives, and the lower right entry the number of true negatives. The off-diagonal entries correspond to errors, with the upper right entry the number of false negatives, and the lower left entry the number of false positives. The total of the four numbers is $N$, the number of examples in the data set. Sometimes the totals of the rows and columns are shown, as in the example above. Various performance metrics are expressed in terms of the numbers in the confusion matrix. The *error rate* is the total number of errors (of both kinds) divided by the total number of examples, *i.e.*, $(N_{\rm fp} + N_{\rm fn})/N$. The *true positive rate* (also known as the *sensitivity* or *recall rate*) is $N_{\rm tp}/N_{\rm p}$. The *false positive rate* (also known as the *false alarm rate*) is $N_{\rm fp}/N_{\rm n}$. (The false positive rate is also sometimes given as the *specificity*, which is one minus the false positive rate.)

An example confusion matrix is given below for the performance of a spam detector on a data set of $N = 1266$ examples (emails) of which 127 are SPAM ($y = +1$) and the remaining 1139 are NOT SPAM ($y = -1$).

| | $\hat{y} = +1$ | $\hat{y} = -1$ | total |
|---|---|---|---|
| $y = +1$ | 95 | 32 | 127 |
| $y = -1$ | 19 | 1120 | 1139 |
| total | 114 | 1152 | 1266 |

On the data set, this classifier has 95 true positives and 1120 true negatives, 19 false positives, and 32 false negatives. Its error rate is $(19+32)/1266 = 4.03\%$. Its true positive rate is $95/127 = 74.8\%$ (meaning it is detecting around 75% of the spam in the data set), and its false positive rate is $19/1139 = 1.67\%$ (meaning it incorrectly labeled around 1.7% of the non-spam message as spam).

### 8.4.2 Least-squares classifier

Our goal is to find a model $\hat{f}$ that is consistent with the data, which in this context means has a small error rate. (As in real-valued data fitting, there is another goal, which we discuss in §8.5.) Many sophisticated methods have been developed for constructing a Boolean model or classifier from a data set. *Logistic regression* and *support vector machine* are two methods that are widely used. But here we discuss a very simple method, based on least-squares, that can work quite well, though not as well as the more sophisticated methods.

We first carry out ordinary real-valued least-squares fitting of the outcome, ignoring for the moment that the outcome $y$ takes on only the values $-1$ and $+1$. We choose basis functions $f_1, \ldots, f_p$, and then choose the parameters $\theta_1, \ldots, \theta_p$ so as to minimize the sum squared error

$$(\tilde{f}(x_1) - y_1)^2 + \cdots + (\tilde{f}(x_N) - y_N)^2,$$

where $\tilde{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x)$. We use the notation $\tilde{f}$, since this function is not our final model $\hat{f}$. The function $\tilde{f}$ is the least-squares fit over our data set, and $\tilde{f}(x)$, for a general vector $x$, is a number.

Our final classifier is then taken to be

$$\hat{f}(x) = \mathbf{sign}(\tilde{f}(x)), \qquad (8.10)$$

where $\mathbf{sign}(a) = +1$ for $a \geq 0$ and $-1$ for $a < 0$. We call this classifier the *least-squares classifier.*

The intuition behind the least-squares classifier is simple. The value $\tilde{f}(x_i)$ is a number, which (ideally) is near $+1$ when $y_i = +1$, and near $-1$ when $y_i = -1$. If we are forced to guess one of the two possible outcomes $+1$ or $-1$, it is natural to choose $\mathbf{sign}(\tilde{f}(x_i))$. Intuition suggests that the number $\tilde{f}(x_i)$ can be related to our confidence in our guess $\hat{y}_i = \mathbf{sign}(\tilde{f}(x_i))$: When $\tilde{f}(x_i)$ is near 1 we have confidence in our guess $\hat{y}_i = +1$; when it is small and negative (say, $\tilde{f}(x_i) = -0.03$), we guess $\hat{y}_i = -1$, but our confidence in the guess will be low. We won't pursue this idea further in this book.

The least-squares classifier is often used with a regression model, *i.e.*, $\tilde{f}(x) = x^T\beta + v$, in which case the classifier has the form

$$\hat{f}(x) = \mathbf{sign}(x^T\beta + v).$$

We can easily interpret the coefficients in this model. For example, if $\beta_7$ is negative, it means that the larger the value of $x_7$ is, the more likely we are to guess $\hat{y} = -1$. If $\beta_4$ is the coefficient with the largest magnitude, then we can say that $x_4$ is the feature that contributes the most to our classification decision.

**Variations.** There are many variations on the basic least-squares classifier described above. One useful modification of (8.10) is to skew the decision point, by adding a shift before taking the sign:

$$\hat{f}(x) = \mathbf{sign}(\tilde{f}(x) - \alpha), \qquad (8.11)$$

where $\alpha$ is a parameter that we choose. The classifier is then

$$\hat{f}(x) = \left\{ \begin{array}{ll} +1 & \tilde{f}(x) \geq \alpha \\ -1 & \tilde{f}(x) < \alpha. \end{array} \right.$$

By choosing $\alpha$ above zero, we make the guess $\hat{f}(x) = +1$ less frequently, so we make fewer false positive errors, and more false negative errors. This can be useful if we care more about one type of error (say, false negatives) than the other.

By sweeping $\alpha$ over a range, we obtain a family of classifiers that vary in their false positive and false negative rates. A common plot, with the strange name *receiver operating characteristic* (ROC), shows the true positive rate on the vertical axis and false positive rate on the horizontal axis. (The name comes from radar systems during World War II, where $y = +1$ meant that an enemy vehicle or ship was present, and $\hat{y} = +1$ means that an enemy vehicle was detected.)

### 8.4.3 Multi-class classifiers

The idea behind the least-squares Boolean classifier can be extended to handle classification problems with $K > 2$ labels or possible outcomes. For each possible label value, we construct a new data set with the Boolean label $+1$ if the label has the given value, and $-1$ otherwise. (This is sometimes called a *one-versus-others* classifier.) From these $K$ Boolean classifiers we must create a classifier that chooses one of the $K$ possible labels. We do this by selecting the label for which the least-squares regression fit has the highest value, which roughly speaking is the one with the highest level of confidence. Our classifier is then

$$\hat{f}(x) = \underset{\ell \in \mathcal{L}}{\operatorname{argmax}}\, \tilde{f}^{\ell}(x),$$

where $\tilde{f}^{\ell}$ is the least-squares regression model for label $\ell$ against the others, and $\mathcal{L}$ is the set of possible labels. The notation argmax means the index of the largest value among the numbers $\tilde{f}^{\ell}(x)$, for $\ell \in \mathcal{L}$.

## 8.5 Validation

**Generalization ability.** In this section we address a key point in model fitting: The goal of model fitting is typically *not* to just achieve a good fit on the given data set, but rather to achieve a good fit on *new data that we have not yet seen.* This leads us to a basic question: How well can we expect a model to predict $y$ for future or other unknown values of $x$? Without some assumptions about the future data, there is no good way to answer this question.

One very common assumption is that the data are described by a formal probability model. With this assumption, techniques from probability and statistics can be used to predict how well a model will work on new, unseen data. This approach has been very successful in many applications, and we hope that you will learn about these methods in another course. In this book, however, we will take a simple intuitive approach to this issue.

The ability of a model to predict the outcomes for new unseen data values is called its *generalization ability.* If it predicts outcomes well, it is said to have good generalization ability; in the opposite case, it is said to have poor generalization ability. So our question is: How can we assess the generalization ability of a model?

**Validation on a test set.** A simple but effective method for assessing the generalization ability of a model is called *out-of-sample validation.* We divide the data we have into two sets: a *training set* and a *test set* (also called a *validation set*). This is often done randomly, with 80% of the data put into the training set and 20% put into the test set. Another common choice for the split ratio is 90%–10%. A common way to describe this is to say that '20% of the data were reserved for validation'.

To fit our model, we use *only the data in the training set.* The model that we come up with is based only on the data in the training set; the data in the test set has never been 'seen' by the model. Then we judge the model by its RMS fit *on the test set.* Since the model was developed without any knowledge of the test set data, the test data is effectively data that are new and unseen, and the performance of our model on this data gives us at least an idea of how our model will perform on new, unseen data. If the RMS prediction error on the test set is large, then we can conclude that our model has poor generalization ability. Assuming that the test data is 'typical' of future data, the RMS prediction error on the test set is what we might guess our RMS prediction error will be on new data.

If the RMS prediction error of the model on the training set is similar to the RMS prediction error on the test set, we have increased confidence that our model has reasonable generalization ability. (A more sophisticated validation method called *cross-validation*, described below, can be used to gain even more confidence.)

For example, if our model achieves an RMS prediction error of 10% (compared to $\mathbf{rms}(y)$) on the training set and 11% on the training set, we can *guess* that it will have a similar RMS prediction error on other unseen data. But there is no guarantee of this, without further assumptions about the data. The basic assumption we are making here is that the future data will 'look like' the test data, or that the test data were 'typical'. Ideas from Statistics can make this idea more precise, but we will leave this idea informal and intuitive.

**Over-fitting.** When the RMS prediction error on the training set is much smaller than the RMS prediction error on the test set, we say that the model is *over-fit.* It tells us that, for the purposes of making predictions on new, unseen data, the model is much less valuable than its performance on the training data suggests.

Good models, which perform well on new data, do not suffer from over-fit. Roughly speaking, an over-fit model trusts the data it has seen (*i.e.*, the training set) too much; it is too sensitive to the changes in the data that will likely be seen in the future data. We will describe below several methods for avoiding over-fit.

**Choosing among different models.** We can use least-squares fitting to fit multiple models to the same data. For example, in univariate fitting, we can fit a constant, an affine function, a quadratic, or a higher order polynomial. Which is the best model among these? Assuming that the goal is to make good predictions on new, unseen data, *we should choose the model with the smallest RMS prediction error on the test set.* Since the RMS prediction error on the test set is only a guess about what we might expect for performance on new, unseen data, we can soften this advice to *we should choose a model that has test set RMS error that is near the minimum over the candidates.*

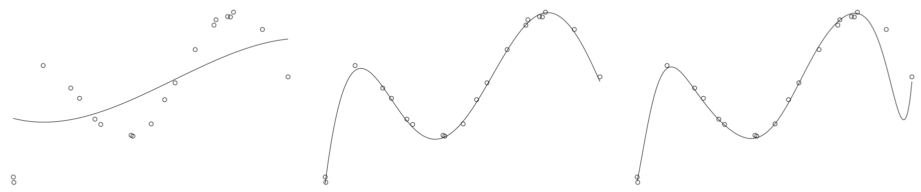

**Figure 8.3** The polynomials fits of figure 8.2 evaluated on a test set of 20 points.

We observed earlier that when we add basis functions to a model, our fitting error on the training data can only decrease (or stay the same). But this is not true for the test error. The test error need not decrease when we add more basis functions. Indeed, when we have too many basis functions, we can expect over-fit, *i.e.*, larger prediction error.

If we have a sequence of basis functions $f_1, f_2, \ldots$, we can consider models based on using just $f_1$ (which is typically the constant function 1), then $f_1$ and $f_2$, and so on. As we increase $p$, the number of basis functions, our training error will go down (or stay the same). But the test error typically decreases, and then starts to increase when $p$ is too large, and the resulting models suffer from over-fit. The intuition for this typical behavior is that for $p$ too small, our model is 'too simple' to fit the data well, and so cannot make good predictions; when $p$ is too large, our model is 'too complex' and suffers from over-fit, and so makes poor predictions. Somewhere in the middle, where the model achieves near minimum test set performance, is a good (or several good) choices of $p$.

To illustrate these ideas, we consider the example shown in figure 8.2. Using a training set of 20 points, we find least-squares fits of polynomials of degrees $0, 1, \ldots, 10$. (The polynomial fits of degrees 3, 5, and 10 are shown in the figure.) We now obtain a new set of data for validation, with 20 points. These test data are plotted along with the polynomial fits obtained from the training data in figure 8.3. This is a real check of our models, since these data points were not used to develop the models.

Figure 8.4 shows the RMS training and test errors for polynomial fits of different degrees. We can see that the RMS training error decreases with every increase in degree. The RMS test error decreases until degree 5 (with a small increase in going from degree 1 to degree 2), and starts to increase for degrees larger than 5. This plot suggests that a polynomial fit of degree 5 (or possibly 6) is a reasonable choice.

With a 5th order polynomial, the RMS test error for both training and test sets is around 0.06. It is a good sign, in terms of generalization ability, that the training and test errors are similar. While there are no guarantees, we can guess that the 5th order polynomial model will have an RMS error around 0.06 on new, unseen data, provided the new, unseen data is sufficiently similar to the test set data.

**Cross validation.** Cross-validation is an extension of out-of-sample validation that can be used to get even more confidence in the generalization ability of a model.

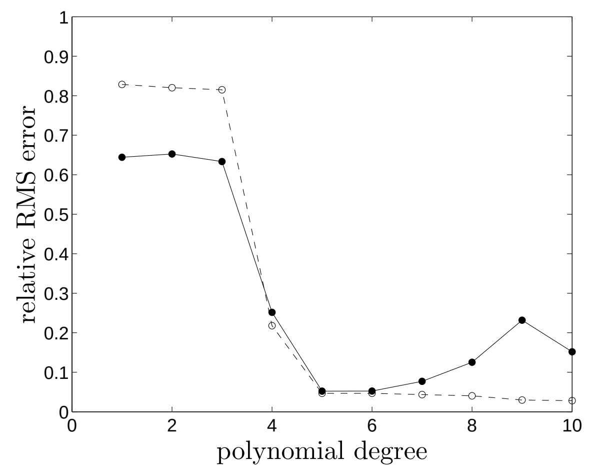

**Figure 8.4** RMS error versus polynomial degree for the fitting example in figures 8.2 and 8.3. Open circles indicate RMS errors on the data set. Filled circles show RMS errors on the test set.

We divide the original data set into 10 sets, called *folds*. We then fit the model using folds $1, 2, \ldots, 9$ as training data, and fold 10 as test data. (So far, this is the same as out-of-sample validation.) We then fit the model using folds $1, 2, \ldots, 8, 10$ as training data and fold 9 as the test data. We continue, fitting a model for each choice of one of the folds as the test set. (This is also called *leave-one-out* validation.) We end up with 10 (presumably different) models, and 10 assessments of these models using the fold that was not used to fit the model. If the test fitting performance of these 10 models is similar, then we expect the same, or at least similar, performance on new unseen data. (We have described 10-fold cross-validation here; 5-fold validation is also commonly used.)

**Validation in classification problems.** In classification problems we are concerned with the error rate, and not the RMS fit. So out-of-sample validation and cross-validation are carried out using the performance that we care about, *i.e.*, the error rate. We may care more about one of the errors (in Boolean classification) than the other; in multi-class classification, we may care more about some errors in the confusion matrix than others.

# Chapter 9

# Multi-objective least-squares

In this chapter we consider the problem of choosing a vector that achieves a compromise in making two or more norm squared objectives small. The idea is widely used in data fitting, image reconstruction, control, and other applications.

## 9.1 Multi-objective least-squares

In the basic least-squares problem (8.1), we seek the vector $\hat{x}$ that minimizes the single objective function $\|Ax-b\|^2$. In some applications we have *multiple* objectives, all of which we would like to be small:

$$J_1 = \|A_1x - b_1\|^2, \quad \ldots, \quad J_k = \|A_kx - b_k\|^2.$$

Here $A_i$ is an $m_i \times n$ matrix, and $b_i$ is an $m_i$-vector. We can use least-squares to find the $x$ that makes any one of these objectives as small as possible (provided the associated matrix has linearly independent columns). This will give us (in general) $k$ different least-squares approximate solutions. But we seek a *single* $\hat{x}$ that gives a compromise, and makes them all small, to the extent possible. We call this the *multi-objective* (or *multi-criterion*) least-squares problem, and refer to $J_1, \ldots, J_k$ as the $k$ objectives.

**Multi-objective least-squares via weighted sum.** A standard method for finding a value of $x$ that gives a compromise in making all the objectives small is to choose $x$ to minimize a *weighted sum objective*:

$$J = \lambda_1 J_1 + \cdots + \lambda_k J_k = \lambda_1\|A_1x - b_1\|^2 + \cdots + \lambda_k\|A_kx - b_k\|^2, \qquad (9.1)$$

where $\lambda_1, \ldots, \lambda_k$ are positive *weights*, that express our relative desire for the terms to be small. If we choose all $\lambda_i$ to be one, the weighted-sum objective is the sum of the objective terms; we give each of them equal weight. If $\lambda_2$ is twice as large as $\lambda_1$, it means that we attach twice as much weight to the objective $J_2$ as to $J_1$. Roughly speaking, we care twice as strongly that $J_2$ should be small, compared to our desire that $J_1$ should be small. We will discuss later how to choose these weights.

Scaling all the weights in the weighted-sum objective (9.1) by any positive number is the same as scaling the weighted-sum objective $J$ by the number, which does not change its minimizers. Since we can scale the weights by any positive number, it is common to choose $\lambda_1 = 1$. This makes the first objective term $J_1$ our *primary* objective; we can interpret the other weights as being relative to the primary objective.

**Weighted sum least-squares via stacking.** We can minimize the weighted-sum objective function (9.1) by expressing it as a standard least-squares problem. We start by expressing $J$ as the norm-squared of a single vector:

$$J = \left\| \begin{bmatrix} \sqrt{\lambda_1}(A_1 x - b_1) \\ \vdots \\ \sqrt{\lambda_k}(A_k x - b_k) \end{bmatrix} \right\|^2,$$

where we use $\|(a_1, \ldots, a_k)\|^2 = \|a_1\|^2 + \cdots + \|a_k\|^2$ for any vectors $a_1, \ldots, a_k$. So we have

$$J = \left\| \begin{bmatrix} \sqrt{\lambda_1} A_1 \\ \vdots \\ \sqrt{\lambda_k} A_k \end{bmatrix} x - \begin{bmatrix} \sqrt{\lambda_1} b_1 \\ \vdots \\ \sqrt{\lambda_k} b_k \end{bmatrix} \right\|^2 = \left\| \tilde{A}x - \tilde{b} \right\|^2,$$

where $\tilde{A}$ and $\tilde{b}$ are the matrix and vector

$$\tilde{A} = \begin{bmatrix} \sqrt{\lambda_1} A_1 \\ \vdots \\ \sqrt{\lambda_k} A_k \end{bmatrix}, \qquad \tilde{b} = \begin{bmatrix} \sqrt{\lambda_1} b_1 \\ \vdots \\ \sqrt{\lambda_k} b_k \end{bmatrix}. \tag{9.2}$$

The matrix $\tilde{A}$ is $m \times n$, and the vector $\tilde{b}$ has length $m$, where $m = m_1 + \cdots + m_k$.

We have now reduced the problem of minimizing the weighted-sum least-squares objective to a standard least-squares problem. Provided the columns of $\tilde{A}$ are independent, the minimizer is unique, and given by

$$\begin{aligned} \hat{x} &= (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T \tilde{b} \\ &= (\lambda_1 A_1^T A_1 + \cdots + \lambda_k A_k^T A_k)^{-1} (\lambda_1 A_1^T b_1 + \cdots + \lambda_k A_k^T b_k). \end{aligned}$$

This reduces to our standard formula for the solution of a least-squares problem when $k = 1$ and $\lambda_1 = 1$. (In fact, when $k = 1$, $\lambda_1$ does not matter.) We can compute $\hat{x}$ via the QR factorization of $\tilde{A}$.

**Independent columns of stacked matrix.** Our assumption (8.2) that the columns of $\tilde{A}$ in (9.2) are independent is not the same as assuming that each of $A_1, \ldots, A_k$ has independent columns. We can state the condition that $\tilde{A}$ has independent columns as: There is no nonzero vector $x$ that satisfies $A_i x = 0$, $i = 1, \ldots, k$. This implies that if just *one* of the matrices $A_1, \ldots, A_k$ has independent columns, then $\tilde{A}$ does.

The stacked matrix $\tilde{A}$ can have independent columns even when none of the matrices $A_1, \ldots, A_k$ do. This can happen when $m_i < n$ for all $i$, *i.e.*, all $A_i$ are wide. However, we must have $m_1 + \cdots + m_k \geq n$, since $\tilde{A}$ must be tall (or square) for the independent columns assumption to hold.

**Optimal trade-off curve.** We start with the special case of two objectives (also called the *bi-criterion problem*), and write the weighted-sum objective as

$$J = J_1 + \lambda J_2 = \|A_1x - b_1\|^2 + \lambda \|A_2x - b_2\|^2,$$

where $\lambda > 0$ is the relative weight put on the second objective, compared to the first. For small $\lambda$, we care much more about $J_1$ being small than $J_2$ being small; for $\lambda$ large, we care much less about $J_1$ being small than $J_2$ being small.

Let $\hat{x}(\lambda)$ denote the weighted-sum least-squares solution $\hat{x}$ as a function of $\lambda$, assuming the stacked matrices have independent columns. These points are called *Pareto optimal*, which means there is no point $z$ that satisfies

$$\|A_1z - b_1\|^2 \leq \|A_1\hat{x}(\lambda) - b_1\|^2, \qquad \|A_2z - b_2\|^2 \leq \|A_2\hat{x}(\lambda) - b_2\|^2,$$

with one of the inequalities holding strictly. Roughly speaking, there is no point $z$ that is as good as $\hat{x}(\lambda)$ in one of the objectives, and beats it on the other one. To see why this is the case, we note that any such $z$ would have a value of $J$ that is less than that achieved by $\hat{x}(\lambda)$, which minimizes $J$, a contradiction.

We can plot the two objectives $\|A_1\hat{x}(\lambda) - b_1\|^2$ and $\|A_2\hat{x}(\lambda) - b_2\|^2$ against each other, as $\lambda$ varies over $(0, \infty)$, to understand the trade-off of the two objectives. This curve is called the *optimal trade-off curve* of the two objectives. There is no point $z$ that achieves values of $J_1$ and $J_2$ that lies below and to the left of the optimal trade-off curve.

**Simple example.** We consider a simple example with two objectives, with $A_1$ and $A_2$ both $10 \times 5$ matrices. The entries of the weighted least-squares solution $\hat{x}(\lambda)$ are plotted against $\lambda$ in figure 9.1. On the left, where $\lambda$ is small, $\hat{x}(\lambda)$ is very close to the least-squares approximate solution for $A_1, b_1$. On the right, where $\lambda$ is large, $\hat{x}(\lambda)$ is very close to the least-squares approximate solution for $A_2, b_2$. In between the behavior of $\hat{x}(\lambda)$ is very interesting; for instance, we can see that $\hat{x}(\lambda)_3$ first increases with increasing $\lambda$ before eventually decreasing.

Figure 9.2 shows the values of the two objectives $J_1$ and $J_2$ versus $\lambda$. As expected, $J_1$ increases as $\lambda$ increases, and $J_2$ decreases as $\lambda$ increases. (It can be shown that this always holds.) Roughly speaking, as $\lambda$ increases we put more emphasis on making $J_2$ small, which comes at the expense of making $J_1$ bigger. The optimal trade-off curve for this bi-criterion problem is plotted in figure 9.3. The left end-point corresponds to minimizing $\|A_1x - b_1\|^2$, and the right end-point corresponds to minimizing $\|A_2x - b_2\|^2$. We can conclude, for example, that there is no vector $z$ that achieves $\|A_1z - b_1\|^2 \leq 3$ and $\|A_2z - b_2\|^2 \leq 2$.

The steep slope of the optimal trade-off curve near the left end-point means that we can achieve a substantial reduction in $J_2$ with only a small increase in $J_1$. The small slope of the optimal trade-off curve near the right end-point means that we can achieve a substantial reduction in $J_1$ with only a small increase in $J_2$. This is quite typical, and indeed, is why multi-criterion least-squares is useful.

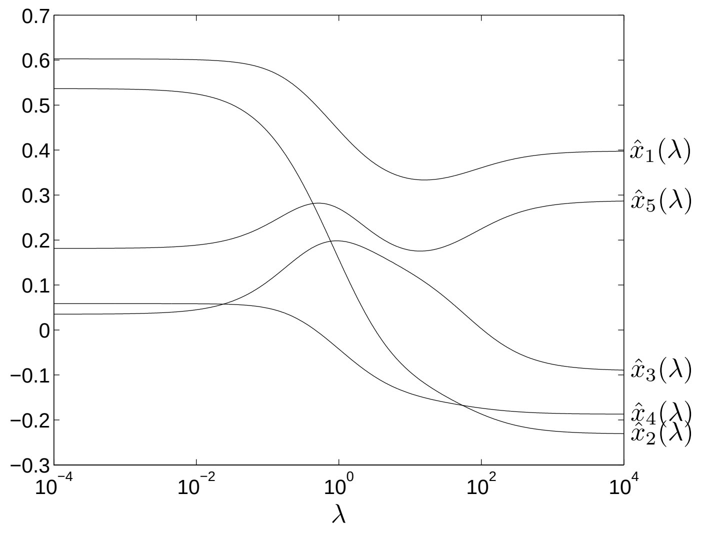

**Figure 9.1** Weighted-sum least-squares solution $\hat{x}(\lambda)$ as a function of $\lambda$ for a bi-criterion least-squares problem with five variables.

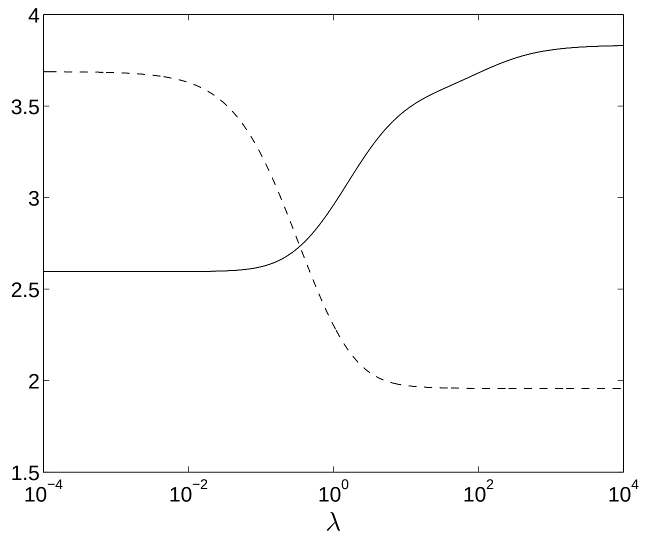

**Figure 9.2** Objective functions $J_1 = \|A_1\hat{x}(\lambda) - b_1\|^2$ (solid line) and $J_2 = \|A_2\hat{x}(\lambda) - b_2\|^2$ (dashed line) as functions of $\lambda$ for the bi-criterion problem in figure 9.1.

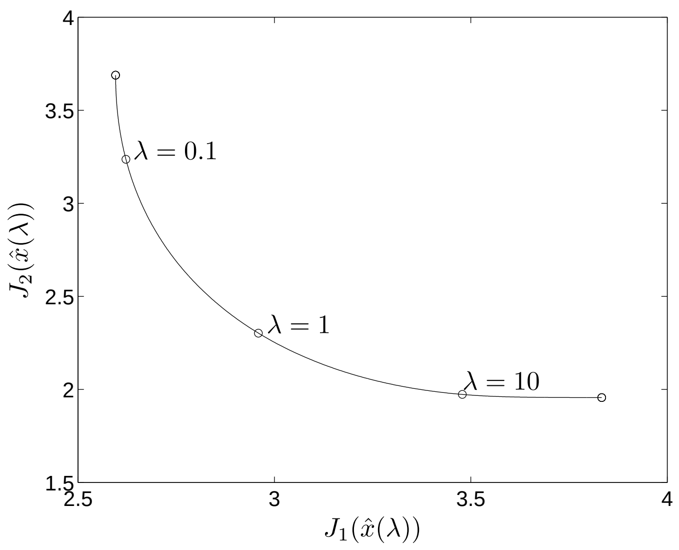

**Figure 9.3** Optimal trade-off curve for the bi-criterion least-squares problem of figures 9.1 and 9.2.

**Optimal trade-off surface.** Above we described the case with $k = 2$ objectives. When we have more than 2 objectives, the interpretation is similar, although it is harder to plot the objectives, or the values of $\hat{x}$, versus the weights. For example with $k = 3$ objectives, we have two weights, $\lambda_2$ and $\lambda_3$, which give the relative weight of $J_2$ and $J_3$ compared to $J_1$. Any solution $\hat{x}(\lambda)$ of the weighted least-squares problem is Pareto optimal, which means that there is no point that achieves values of $J_1, J_2, J_3$ less than or equal to those obtained by $\hat{x}(\lambda)$, with strict inequality holding for at least one of them. As the parameters $\lambda_2$ and $\lambda_3$ vary over $(0, \infty)$, the values of $J_1, J_2, J_3$ sweep out the *optimal trade-off surface*.

**Using multi-objective least-squares.** In the rest of this chapter we will see several specific applications of multi-objective least-squares. Here we give some general remarks on how it is used in applications.

First we identify a primary objective $J_1$ that we would like to be small. The objective $J_1$ is typically the one that would be used in an ordinary single-objective least-squares approach, such as the mean square error of a model on some training data, or the mean-square deviation from some target or goal.

We also identify one or more *secondary objectives* $J_2,\ J_3, \ldots, J_k$, that we would also like to be small. These secondary objectives are typically generic ones, like the desire that some parameters be 'small' or 'smooth', or close to some previous or prior value. In estimation applications these secondary objectives typically correspond to some kind of prior knowledge or assumption about the vector $x$ that we seek. We wish to minimize our primary objective, but are willing to accept an increase in it, if this gives a sufficient decrease in the secondary objectives.

The weights are treated like 'knobs' in our method, that we change ('turn' or 'tune' or 'tweak') to achieve a value of $\hat{x}$ that we like (or can live with). For given candidate values of $\lambda$ we evaluate the objectives; if we decide that $J_2$ is larger than we would like, but we can tolerate a somewhat larger $J_3$, then we increase $\lambda_2$ and decrease $\lambda_3$, and find $\hat{x}$ and the associated values of $J_1, J_2, J_3$ using the new weights. This is repeated until a reasonable trade-off among them has been obtained. In some cases we can be principled in how we adjust the weights; for example, in data fitting, we can use validation to help guide us in the choice of the weights. In many other applications, it comes down to (application-specific) judgment or even taste.

The additional terms $\lambda_2 J_2, \ldots, \lambda_k J_k$ that we add to the primary objective $J_1$, are sometimes called *regularization (terms)*. The secondary objectives are the sometimes described by name, as in 'least-squares fitting with smoothness regularization'.

## 9.2 Control

In control applications, the goal is to decide on a set of actions or inputs, specified by an $n$-vector $x$, that achieve some goals. The actions result in some outputs or effects, given by an $m$-vector $y$. We consider here the case when the inputs and outputs are related by an affine model

$$y = Ax + b.$$

The $m \times n$ matrix $A$ and $m$-vector $b$ characterize the *input-output mapping* of the system. The model parameters $A$ and $b$ are found from analytical models, experiments, computer simulations, or fit to past (observed) data. Typically the input or action $x = 0$ has some special meaning. The $m$-vector $b$ gives the output when the input is zero. In many cases the vectors $x$ and $y$ represent *deviations* of the inputs and outputs from some standard values.

We typically have a desired or target output, denoted by the $m$-vector $y^{\mathrm{des}}$. The primary objective is

$$J_1 = \|Ax + b - y^{\mathrm{des}}\|^2,$$

the norm squared deviation of the output from the desired output. The main objective is to choose an action $x$ so that the output is as close as possible to the desired value.

There are many possible secondary objectives. The simplest one is the norm squared value of the input, $J_2 = \|x\|^2$, so the problem is to optimally trade off missing the target output (measured by $\|y - y^{\mathrm{des}}\|^2$), and keeping the input small (measured by $\|x\|^2$).

Another common secondary objective has the form $J_2 = \|x - x^{\mathrm{nom}}\|^2$, where $x^{\mathrm{nom}}$ is a nominal or standard value for the input. In this case the secondary objective it to keep the input close to the nominal value. This objective is sometimes used when $x$ represents a new choice for the input, and $x^{\mathrm{nom}}$ is the current value. In this case the goal is to get the output near its target, while not changing the input much from its current value.

**Control of heating and cooling.** As an example, $x$ could give the vector of $n$ heating (or cooling) power levels in a commercial building with $n$ air handling units (with $x_i > 0$ meaning heating and $x_i < 0$ meaning cooling) and $y$ could represent the resulting temperature at $m$ locations in the building. The matrix $A$ captures the effect of each of $n$ heating/cooling units on the temperatures in the building at each of $m$ locations; the vector $b$ gives the temperatures at the $m$ locations when no heating or cooling is applied. The desired or target output might be $y^{\text{des}} = T^{\text{des}}\mathbf{1}$, assuming the target temperature is the same at all locations. The primary objective $\|y - y^{\text{des}}\|^2$ is the sum of squares of the deviations of the location temperatures from the target temperature. The secondary objective $J_2 = \|x\|^2$, the norm squared of the vector of heating/cooling powers, would be reasonable, since it is at least roughly related to the energy cost of the heating and cooling.

We find tentative choices of the input by minimizing $J_1 + \lambda_2 J_2$ for various values of $\lambda_2$. If for the current value of $\lambda_2$ the heating/cooling powers are larger than we'd like, we increase $\lambda_2$ and re-compute $\hat{x}$.

**Dynamics.** The system can also be dynamic, meaning that we take into account time variation of the input and output. In the simplest case $x$ is the time series of a scalar input, so $x_i$ is the action taken in period $i$, and $y_i$ is the (scalar) output in period $i$. In this setting, $y^{\text{des}}$ is a desired trajectory for the output. A very common model for modeling dynamic systems, with $x$ and $y$ representing scalar input and output time series, is a convolution: $y = h * x$. In this case, $A$ is Toeplitz, and $b$ represents a time series, which is that the output would be with $x = 0$.

As a typical example in this category, the input $x_i$ can represent the torque applied to the drive wheels of a locomotive (say, over 1 second intervals), and $y_i$ is the locomotive speed.

In addition to the usual secondary objective $J_2 = \|x\|^2$, it is common to have an objective that the input should be smooth, *i.e.*, not vary too rapidly over time. This is achieved with the objective $\|Dx\|^2$, where $D$ is the $(n-1) \times n$ first difference matrix.

## 9.3 Estimation and inversion

In the broad application area of *estimation* (also called *inversion*), the goal is to estimate a set of $n$ values (also called parameters), the entries of the $n$-vector $x$. We are given a set of $m$ *measurements*, the entries of an $m$-vector $y$. The parameters and measurements are related by

$$y = Ax + v,$$

where $A$ is a known $m \times n$ matrix, and $v$ is an unknown $m$-vector. The matrix $A$ describes how the measured values (*i.e.*, $y_i$) depend on the unknown parameters (*i.e.*, $x_j$). The $m$-vector $v$ is the *measurement error* or *measurement noise*, and is unknown but presumed to be small. The estimation problem is to make a sensible guess as to what $x$ is, given $y$ (and $A$), and prior knowledge about $x$.

If the measurement noise were zero, and $A$ has independent columns, we could recover $x$ exactly, using $x = A^\dagger y$. (This is called *exact inversion*.) Our job here is to *guess* $x$, even when these strong assumptions do not hold. Of course we cannot expect to find $x$ exactly, when the measurement noise is nonzero, or when $A$ does not have independent columns. This is called *approximate inversion*, or in some contexts, just *inversion*.

The matrix $A$ can be wide, square, or tall; the same methods are used to estimate $x$ in all three cases. When $A$ is wide, we would not have enough measurements to determine $x$ from $y$, even without the noise (*i.e.*, with $v = 0$). In this case we have to also rely on our prior information about $x$ to make a reasonable guess. When $A$ is square or tall, we would have enough measurements to determine $x$, if there were no noise present. Even in this case, judicious use of multiple-objective least-squares can incorporate our prior knowledge in the estimation, and yield far better results.

If we guess that $x$ has the value $\hat{x}$, then we are implicitly making the guess that $v$ has the value $y - A\hat{x}$. If we assume that smaller values of $v$ (measured by $\|v\|$) are more plausible than larger values, then a sensible choice for $\hat{x}$ is the least-squares approximate solution, which minimizes $\|A\hat{x} - y\|^2$. We will take this as our primary objective.

Our prior information about $x$ about enters in one or more secondary objectives. Simple examples are listed below.

- $\|x\|^2$: $x$ should be small.
- $\|x - x^{\text{tar}}\|^2$: $x$ should be near $x^{\text{tar}}$.
- $\|Dx\|^2$, where $D$ is the first difference matrix: $x$ should smooth, *i.e.*, $x_{i+1}$ should be near $x_i$.

We will see other regularizers below.

Finally, we will choose our estimate $\hat{x}$ by minimizing

$$\|Ax - y\|^2 + \lambda_2 J_2(x) + \cdots + \lambda_p J_p(x),$$

where $\lambda_i > 0$ are weights, and $J_2, \ldots, J_p$ are the regularization terms.

**Tikhonov regularized inversion.** Choosing $\hat{x}$ to minimize

$$\|Ax - y\|^2 + \lambda \|x\|^2$$

for some choice of $\lambda > 0$ is called *Tikhonov regularized inversion*. Here we seek a guess that is consisent with the measurements (*i.e.*, $\|A\hat{x} - y\|^2$ is small), but not too big. The stacked matrix in this case always has independent columns, without any assumption about $A$, which can have any dimensions, and need not have independent columns.

**Equalization.** The vector $x$ represents a transmitted signal or message, consisting of $n$ real values. The matrix $A$ represents the mapping from the transmitted signal to what is received (called the *channel*); $y = Ax + v$ includes noise as well as the action of the channel. Guessing what $x$ is, given $y$, can be thought of as un-doing the effects of the channel. In this context, estimation is called *equalization*.

**Estimating a periodic time series.** Suppose that the $T$-vector $y$ is a (measured) time series, that we believe is a noisy version of a periodic time series, *i.e.*, one that repeats itself every $P$ periods. We might also know or assume that the periodic time series in smooth, *i.e.*, its adjacent values are not too far apart.

Periodicity arises in many time series. For example, we would expect a time series of hourly temperature at some location to approximately repeat itself every 24 hours, or the monthly snowfall in some region to approximately repeat itself every 12 months. (Periodicity with a 24 hour period is called *diurnal*; periodicity with a yearly period is called *seasonal* or *annual*.) As another example, we might expect daily total sales at a restaurant to approximately repeat itself weekly. The goal is to get an estimate of Tuesday's total sales, given some historical daily sales data.

The periodic time series will be represented by a $P$-vector $x$, which gives its values over one period. It corresponds to the full time series

$$\hat{y} = (x, x, \ldots, x)$$

which just repeats $x$, where we assume here for simplicity that $T$ is a multiple of $P$. (If this is not the case, the last $x$ is replaced with a slice of the form $x_{1:k}$.) We can express $\hat{y}$ as $\hat{y} = Ax$, where $A$ is the $T \times P$ selector matrix

$$A = \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix}.$$

Our total square estimation error is $\|Ax - y\|^2$.

We can minimize this objective analytically. The solution $\hat{x}$ is found by averaging the values of $y$ associated with the different entries in $x$. For example, we estimate Tuesday sales by averaging all the entries in $y$ that correspond to Tuesdays. (See exercise **??**.) This simple averaging works well if we have many periods worth of data, *i.e.*, if $T/S$ is large.

A more sophisticated estimate can be found by adding regularization for $x$ to be smooth, based on the assumption that

$$x_1 \approx x_2, \quad \ldots, \quad x_{P-1} \approx x_P, \quad x_P \approx x_1.$$

(Note that we include the 'wrap-around' pair $x_P$ and $x_1$ here.) We measure non-smoothness as $\|D^{\text{circ}}x\|^2$, where $D^{\text{circ}}$ is the $P \times P$ *circular difference matrix*

$$D^{\text{circ}} = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ 1 & & & & -1 \end{bmatrix}.$$

We estimate the periodic time series by minimizing $\|Ax - y\|^2 + \lambda\|D^{\text{circ}}x\|^2$. For $\lambda = 0$ we recover the simple averaging mentioned above; as $\lambda$ gets bigger, the estimated signal becomes smoother, ultimately converging to a constant (which is the mean of the original time series data).

The time series $A\hat{x}$ is called the *extracted seasonal component* of the given time series data $y$ (assuming we are considering yearly variation). Subtracting this from the original data yields the time series $y - A\hat{x}$, which is called the *seasonally adjusted* time series.

The parameter $\lambda$ can be chosen using validation. This can be done by selecting a time interval over which to build the estimate, and another one to validate it. For example, with 4 years of data, we might train our model on the first 3 years of data, and test it on the last year of data.

**Image de-blurring.** The vector $x$ is an image, and the matrix $A$ gives blurring, so $y = Ax + v$ is a blurred, noisy image. Our prior information about $x$ is that it is smooth; neighboring pixels values are not very different from each other. Estimation is the problem of guessing what $x$ is, and is called *de-blurring.*

In least-squares image deblurring we form an estimate $\hat{x}$ by minimizing a cost function of the form

$$\|Ax - y\|^2 + \lambda(\|D_{\rm h}x\|^2 + \|D_{\rm v}x\|^2). \qquad (9.3)$$

Here $D_{\rm v}$ and $D_{\rm h}$ represent vertical and horizontal differencing operations, and the role of the second tem in the weighted sum is to penalize non-smoothness in the reconstructed image. Specifically, suppose the vector $x$ has length $MN$ and contains the pixel intensities of an $M \times N$ image $X$ stored column-wise. Let $D_h$ be the $(N-1)M$ matrix

$$D_{\rm h} = \left[\begin{array}{ccccccc} I & -I & 0 & \cdots & 0 & 0 & 0 \\ 0 & I & -I & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & I & -I & 0 \\ 0 & 0 & 0 & \cdots & 0 & I & -I \end{array}\right],$$

where all blocks have size $M \times M$, and let $D_{\rm v}$ be the $(M-1)N$ matrix

$$D_{\rm v} = \left[\begin{array}{cccc} D & 0 & \cdots & 0 \\ 0 & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D \end{array}\right],$$

where each of the $N$ diagonal blocks $D$ is an $(M-1) \times M$ difference matrix

$$D = \left[\begin{array}{ccccccc} 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \end{array}\right].$$

With these definitions the penalty term in (9.3) is the sum of squared differences of intensities at adjacent pixels in a row or column:

$$\|D_{\rm h}x\|^2 + \|D_{\rm v}x\|^2 = \sum_{i=1}^{M}\sum_{j=1}^{N-1}(X_{ij} - X_{i,j+1})^2 + \sum_{i=1}^{M-1}\sum_{j=1}^{N}(X_{ij} - X_{i+1,j})^2.$$

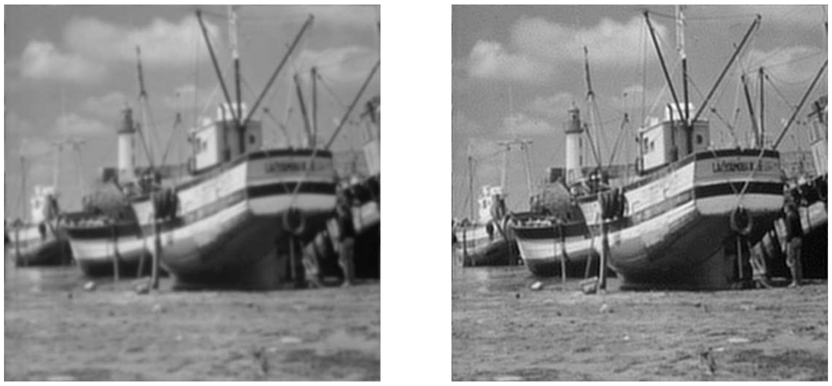

**Figure 9.4** *Left:* Blurred, noisy image. *Right:* Result of regularized least-squares deblurring with $\lambda = 0.007$.

In figures 9.4 and 9.5 we illustrate this this method for an image of size $512 \times 512$. The blurred, noisy image is shown in the left part of figure 9.4. Figure 9.5 shows the estimates $\hat{x}$, obtained by minimizing (9.3), for four different values of the parameter $\lambda$. The best result is obtained for $\lambda$ around 0.007 and is shown on the right in figure 9.4.

## 9.4 Regularized data fitting

We consider least-squares data fitting, as described in §8.3. In §8.5 we considered the issue of over-fitting, where the model performs poorly on new, unseen data, which occurs when the model is too complicated for the given data set. The remedy is to keep the model simple, *e.g.*, by fitting with a polynomial of not too high a degree.

Regularization is another way to avoid over-fitting, different from simply choosing a model that is simple (*i.e.*, does not have too many basis functions). Regularization is also called *de-tuning*, *shrinkage*, or *ridge regression*, for reasons we will explain below.

To motivate regularization, consider the model

$$\hat{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x).$$

We can interpret $\theta_i$ as the amount by which our prediction depends on $f_i(x)$, so if $\theta_i$ is large, our prediction will be very sensitive to changes or variations in the value of $f_i(x)$, such as those we might expect in new, unseen data. This suggests that we should prefer that $\theta_i$ be small, so our model is not too sensitive. There is one exception here: if $\theta_i(x)$ is constant (for example, the number one), then we should not worry about the size of $\theta_i$, since $f_i(x)$ never varies. But we would like all the others to be small, if possible.

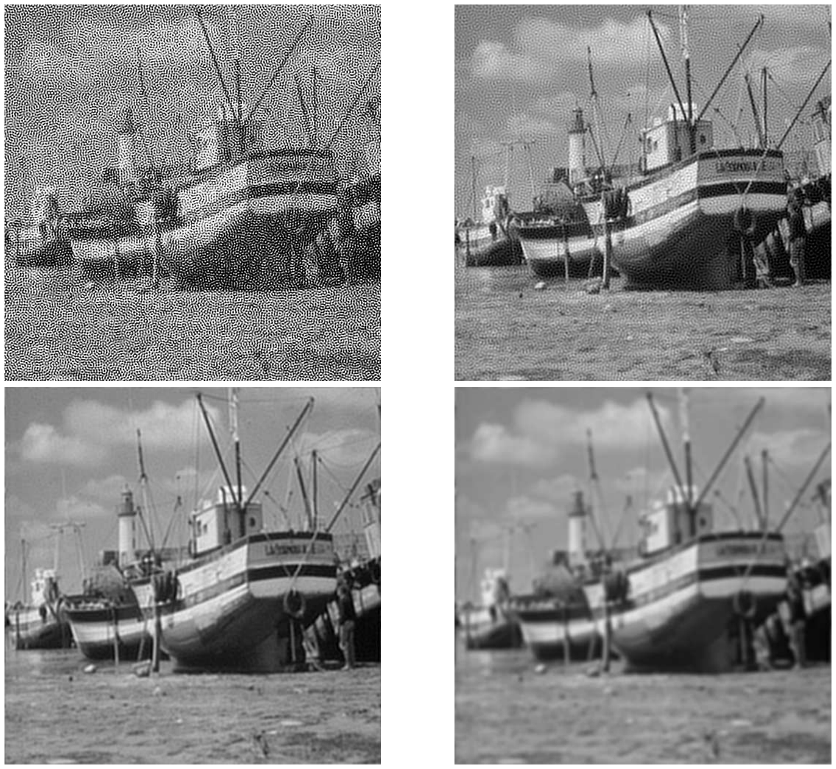

**Figure 9.5** Deblurred images for $\lambda = 10^{-6}$, $10^{-4}$, $10^{-2}$, 1.

This suggests the bi-criterion least-squares problem with primary objective $\|A\theta - y\|^2$, the sum of squares of the prediction errors, and secondary objective $\|\theta_{2:p}\|^2$, assuming that $f_1$ is the constant function one. Thus we should minimize

$$\|A\theta - y\|^2 + \lambda\|\theta_{2:p}\|^2,$$

where $\lambda > 0$ is called the *regularization parameter*.

For the regression model, this weighted objective can be expressed as

$$\|X^T\beta + v\mathbf{1} - y\|^2 + \lambda\|\beta\|^2.$$

Here we penalize $\beta$ being large (because this leads to sensitivity of the model), but not the offset $v$. Choosing $\beta$ to minimize this weighted objective is called *ridge regression*.

**Effect of regularization.** The effect of the regularization is to accept a worse value of sum square fit ($\|A\theta-y\|^2$) in return for a smaller value of $\|\theta_{2:p}\|^2$, which measures the size of the parameters (except $\theta_1$, which is associated with the constant basis function). This explains the name shrinkage: The parameters are smaller than they would be without regularization, *i.e.*, they are shrunk. The term de-tuned suggests that with regularization, the model is not excessively 'tuned' to the training data (which would lead to over-fit).

**Regularization path.** We get a different model for every choice of $\lambda$. The way the parameters change with $\lambda$ is called the *regularization path*. When $p$ is small enough (say, less than 15 or so) the parameter values can be plotted, with $\lambda$ on the horizontal axis. Usually only 30 or 50 values of $\lambda$ are considered, typically spaced logarithmically over a large range.

An appropriate value of $\lambda$ can be chosen via out-of-sample or cross validation. As $\lambda$ increases, the RMS fit on the training data worsens (increases). But (as with model order) the test set RMS prediction error typically decreases as $\lambda$ increases, and then, when $\lambda$ gets too big, it increases. A good choice of regularization parameter is one which approximately minimizes the test set RMS prediction error.

**Independence of columns.** One side benefit of adding regularization to the basic least-squares data fitting method is that the columns of the associated stacked matrix are *always independent*, even if the columns of the matrix $A$ are not. To see this, suppose that

$$\begin{bmatrix} A \\ \sqrt{\lambda}B \end{bmatrix} x = 0,$$

where $B$ is the $(p-1)\times p$ selector matrix

$$B = (e_2^T, \ldots, e_p^T),$$

so $B\theta = \theta_{2:p}$. From the last $p-1$ entries in the equation above, we get $\sqrt{\lambda}x_i = 0$ for $i = 2,\ldots,p$, which implies that $x_2 = \cdots = x_p = 0$. Using these values of $x_2,\ldots,x_p$, and the fact that the first column of $A$ is $\mathbf{1}$, the top $m$ equations become $\mathbf{1}x_1 = 0$, and we conclude that $x_1 = 0$ as well. So the columns of the stacked matrix are always independent.

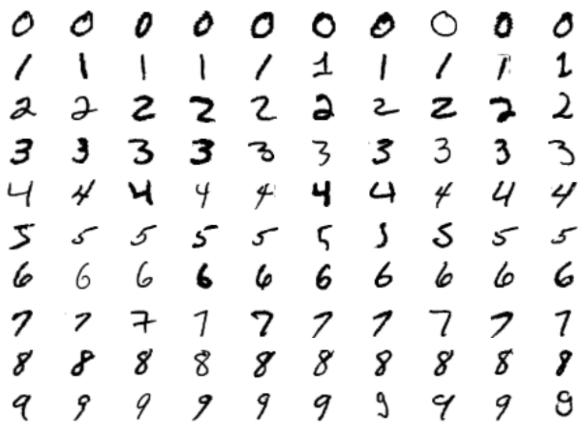
**Figure 9.6** 100 examples from the MNIST database of handwritten digits (yann.lecun.com/exdb/mnist).

**Least-squares classification.** We consider an application of the least-squares multi-class classification method described in section 8.4. The goal of the classifier is to recognize handwritten digits. We use the MNIST data set, which contains 60,000 images of size 28 by 28 (a few samples are shown figure 9.6). The pixel intensities are scaled to lie between 0 and 1. The test set contains $N = 10,000$ images. We add a constant feature, so there are $28^2 + 1 = 785$ features.

For each of the ten digits $k = 0, \ldots, 9$ we compute a binary classifier to distinguish digit $k$ from the other digits. The $k$th classifier has the form $\hat{f}^k(x) = \mathbf{sign}(x^T\beta_k + v_k)$, with coefficients $v_k$, $\beta_k$ computed by solving a regularized least-squares problem

$$\text{minimize} \ \sum_{i=1}^{N}(x_i^T\beta_k + v_k - y_{ik})^2 + \lambda\|\beta_k\|^2. \tag{9.4}$$

Here $y_{ik} = 1$ if training example $x_i$ is an image of digit $k$ and $y_{ik} = -1$ otherwise. We then use the 10 binary classifiers to define a multi-class classifier $\hat{f}(x) = \text{argmax}_{k=0,\ldots,9}(x^T\beta_k + v_k)$.

Figure 9.7 shows the classification error (percentage of misclassified digits) for the training set and test set versus $\lambda$. The classification errors are roughly the same and about 14% for $\lambda \leq 10^3$. With $\lambda = 1000$ the classification error on the test set is 13.5%. The confusion matrix for this classifier is given below.

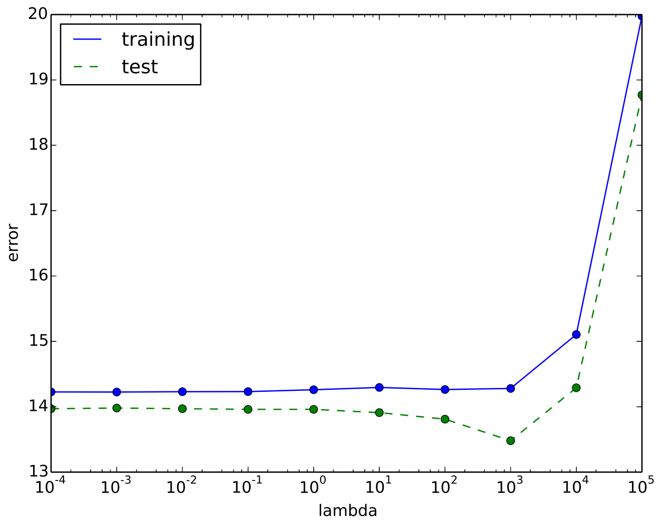

**Figure 9.7** Multiclass classification error in percent versus $\lambda$.

| | Prediction $\hat{\ell}$ | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | total |
| $\ell = 0$ | 940 | 0 | 1 | 2 | 1 | 9 | 18 | 1 | 7 | 1 | 980 |
| $\ell = 1$ | 0 | 1106 | 2 | 2 | 2 | 1 | 5 | 1 | 16 | 0 | 1135 |
| $\ell = 2$ | 17 | 64 | 829 | 19 | 14 | 0 | 23 | 25 | 36 | 5 | 1032 |
| $\ell = 3$ | 5 | 18 | 24 | 885 | 5 | 13 | 9 | 22 | 19 | 10 | 1010 |
| $\ell = 4$ | 1 | 21 | 6 | 0 | 885 | 2 | 10 | 1 | 11 | 45 | 982 |
| $\ell = 5$ | 21 | 16 | 4 | 85 | 28 | 636 | 24 | 20 | 39 | 19 | 892 |
| $\ell = 6$ | 15 | 11 | 4 | 0 | 17 | 16 | 890 | 0 | 5 | 0 | 958 |
| $\ell = 7$ | 5 | 44 | 16 | 5 | 18 | 0 | 1 | 897 | 0 | 42 | 1028 |
| $\ell = 8$ | 13 | 53 | 9 | 32 | 25 | 27 | 19 | 14 | 763 | 19 | 974 |
| $\ell = 9$ | 19 | 13 | 3 | 13 | 67 | 3 | 1 | 63 | 6 | 821 | 1009 |
| total | 1036 | 1346 | 898 | 1043 | 1062 | 707 | 1000 | 1044 | 902 | 962 | 10000 |

## 9.5 Choosing basis functions

# Chapter 10

# Constrained least-squares

In this chapter we discuss a useful extension of the least-squares problem that includes linear equality constraints. Like least-squares problems, the solution of constrained least-squares problems can be reduced to a set of linear equations, and can be computed using the QR factorization. We describe some applications.

## 10.1 Least-squares with equality constraints

In the basic least-squares problem, we seek $x$ that minimizes the objective function $\|Ax - b\|^2$. We now add *constraints* to this problem, by insisting that $x$ satisfy the linear equations $Cx = d$, where the matrix $C$ and the vector $d$ are given. The *linearly constrained least-squares problem* (or just, constrained least-squares problem) is written as

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|^2 \\ \text{subject to} & Cx = d. \end{array} \qquad (10.1)$$

Here $x$, the variable to be found, is an $n$-vector. The problem data (which are given) are the $m \times n$ matrix $A$, the $m$-vector $b$, the $p \times n$ matrix $C$, and the $p$-vector $d$.

We refer to the function $\|Ax - b\|^2$ as the *objective* of the problem, and the set of $p$ linear equality constraints $Cx = d$ as the *constraints* of the problem. An $n$-vector $x$ is called *feasible* (for the problem (10.1)) if it satisfies the constraints, *i.e.*, $Cx = d$. An $n$-vector $\hat{x}$ is called an *optimal point* or *solution* of the optimization problem (10.1) if it is feasible, and if $\|A\hat{x} - b\|^2 \leq \|Ax - b\|^2$ holds for any feasible $x$. In other words, $\hat{x}$ solves the problem (10.1) if it is feasible and has the smallest possible value of the objective function among all feasible vectors.

The constrained least-squares problem combines the problems of solving a set of linear equations (find $x$ that satisfies $Cx = d$) with the least-squares problem (find $x$ that minimizes $\|Ax - b\|^2$). Indeed each of these problems can be considered special cases of the constrained least-squares problem (10.1).

The constrained least-squares problem can also be thought of as a limit of a bi-objective least-squares problem, with primary objective $\|Ax - b\|^2$ and secondary objective $\|Cx - d\|^2$. Roughly speaking, we put infinite weight on the second objective, so that any nonzero value is unacceptable (which forces $x$ to satisfy $Cx = d$). So we would expect (and it can be verified) that minimizing the weighted objective

$$\|Ax - b\|^2 + \lambda\|Cx - d\|^2,$$

for a very large value of $\lambda$ yields a vector close to a solution of the constrained least-squares problem (10.1).

**Least-norm problem.** An important special case of the constrained least-squares problem (10.1) is when $A = I$ and $b = 0$:

$$\begin{array}{ll} \text{minimize} & \|x\|^2 \\ \text{subject to} & Cx = d. \end{array} \tag{10.2}$$

In this problem we seek the vector of least norm that satisfies the linear equations $Cx = d$. For this reason the problem (10.2) is called the *least-norm problem* or *minimum-norm problem*.

## 10.2 Solution of constrained least-squares problem

**Optimality conditions via calculus.** In this section we use calculus to derive the conditions that any solution of the constrained least-squares problem (10.1) must satifsy. (Later we give an independent verification, that does not rely on calculus, that the solution we derive is correct.)

We first write the contrained least-squares problem with the constraints given as a list of $p$ scalar equality constraints:

$$\begin{array}{ll} \text{minimize} & f(x) = \|Ax - b\|^2 \\ \text{subject to} & c_i^T x = d_i, \quad i = 1, \ldots, p, \end{array}$$

where $c_i^T$ are the rows of $C$. To solve this problem we use the *method of Lagrange multipliers*. We form the *Lagrangian function*

$$L(x, z) = f(x) + z_1(c_1^T x - d_1) + \cdots + z_p(c_p^T x - d_p),$$

where $z$ is the $p$-vector of *Lagrange multipliers*. The method of Lagrange multipliers tells us that if $\hat{x}$ is a solution of the constrained least-squares problem, then there is a set of Lagrange multipliers $z$ that satisfy

$$\frac{\partial L}{\partial x_i}(\hat{x}, z) = 0, \quad i = 1, \ldots, n, \qquad \frac{\partial L}{\partial z_i}(\hat{x}, z) = 0, \quad i = 1, \ldots, p. \tag{10.3}$$

These are the optimality conditions for the constrained least-squares problem.

The second set of equations in the optimality conditions can be written as

$$\frac{\partial L}{\partial z_i}(\hat{x}, z) = c_i^T \hat{x} - d_i = 0, \quad i = 1, \ldots, p,$$

which states that $\hat{x}$ satisfies the equality constraints $C\hat{x} = d$ (which we already knew). The first set of equations, however, is more informative. Expanding $f(x)$ out as a sum of terms involving the entries of $x$ (as was done on page 115) and taking the partial derivative of $L$ with respect to $x_i$ we obtain

$$\frac{\partial L}{\partial x_i}(\hat{x}, z) = 2\sum_{j=1}^{n}(A^TA)_{ij}\hat{x}_j - 2(A^Tb)_i + \sum_{j=1}^{p} z_j c_i = 0.$$

We can write these equations in compact matrix-vector form as

$$2(A^TA)\hat{x} - 2A^Tb + C^Tz = 0.$$

Combining this set of linear equations with the feasibility conditions $C\hat{x} = d$, we can write the optimality conditions (10.3) as one set of $m + p$ linear equations in the variables $(\hat{x}, z)$:

$$\left[\begin{array}{cc} 2A^TA & C^T \\ C & 0 \end{array}\right] \left[\begin{array}{c} \hat{x} \\ z \end{array}\right] = \left[\begin{array}{c} 2A^Tb \\ d \end{array}\right]. \qquad (10.4)$$

These equations are called the *KKT equations* for the constrained least-squares problem. (KKT stands for Kharush, Kuhn and Tucker, the names of three researchers who derived the optimality conditions for a much more general form of constrained optimization problem.) The KKT equations (10.4) are an extension of the normal equations (8.4) for a least-squares problem with no constraints. So we have reduced the constrained least-squares problem to the problem of solving a (square) set of $n + p$ linear equations in $n + p$ variables $(\hat{x}, z)$.

**Invertibility of KKT matrix.** The $(m + p) \times (m + p)$ coefficient matrix in (10.4) is called the *KKT matrix*. It is invertible if and only if

$$C \textit{ has independent rows, and } \left[\begin{array}{c} A \\ C \end{array}\right] \textit{ has independent columns.} \qquad (10.5)$$

The first condition requires that $C$ is wide (or square), *i.e.*, that there are fewer constraints than variables. The second condition depends on both $A$ and $C$, and it can be satisfied even when the the columns of $A$ are dependent. The condition (10.5) is the generalization of our assumption (8.2) for unconstrained least-squares (*i.e.*, that $A$ has independent columns).

Before proceeding, let us verify that the KKT matrix is invertible if and only if (10.5) holds. First suppose that the KKT matrix is not invertible. This means that there is a nonzero vector $(\bar{x}, \bar{z})$ with

$$\left[\begin{array}{cc} 2A^TA & C^T \\ C & 0 \end{array}\right] \left[\begin{array}{c} \bar{x} \\ \bar{z} \end{array}\right] = 0.$$

Multiply the top block equation $2A^TA\bar{x} + C^T\bar{z} = 0$ on the left by $\bar{x}^T$ to get

$$2\|A\bar{x}\|^2 + \bar{x}^TC^T\bar{z} = 0.$$

The second block equation, $C\bar{x} = 0$, implies (by taking the transpose) $\bar{x}^T C^T = 0$, so the equation above becomes $2\|A\bar{x}\|^2 = 0$, *i.e.*, $A\bar{x} = 0$. We also have $C\bar{x} = 0$, so

$$\begin{bmatrix} A \\ C \end{bmatrix} \bar{x} = 0.$$

Since the matrix on the left has independent columns (by assumption), we conclude that $\bar{x} = 0$. The second block equation above then becomes $C^T\bar{z} = 0$. But by our assumption that the columns of $C^T$ are independent, we have $\bar{z} = 0$. So $(\bar{x}, \bar{z}) = 0$, which is a contradiction.

The converse is also true. First suppose that the rows of $C$ are dependent. Then there is a nonzero vector $\bar{z}$ with $C^T\bar{z} = 0$. Then

$$\begin{bmatrix} 2A^TA & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \bar{z} \end{bmatrix} = 0,$$

which shows the KKT matrix is not invertible. Now suppose that the stacked matrix in (10.5) has dependent columns, which means there is a nonzero vector $\bar{x}$ for which

$$\begin{bmatrix} A \\ C \end{bmatrix} \bar{x} = 0.$$

Direct calculation shows that

$$\begin{bmatrix} 2A^TA & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \bar{x} \\ 0 \end{bmatrix} = 0,$$

which shows that the KKT matrix is not invertible.

**Solution of constrained least-squares problem.** When the conditions (10.5) hold, the constrained least-squares problem (10.1) has the (unique) solution $\hat{x}$, given by

$$\begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} 2A^TA & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2A^Tb \\ d \end{bmatrix}. \qquad (10.6)$$

(This formula also gives us $z$, the set of Lagrange multipliers.) From (10.6), we observe that the solution $\hat{x}$ is a linear function of $(b, d)$.

**Direct verification of constrained least-squares solution.** We will now show directly, without using calculus, that the solution $\hat{x}$ given in (10.6) is the unique vector that minimizes $\|Ax - b\|^2$ over all $x$ that satisfy the constraints $Cx = d$, when the conditions (10.5) hold. Let $\hat{x}$ and $z$ denote the vectors given in (10.6), so they satisfy

$$2A^TA\hat{x} + C^Tz = 2A^Tb, \qquad C\hat{x} = d.$$

Suppose that $x \neq \hat{x}$ is any vector that satisfies $Cx = d$. We will show that $\|Ax - b\|^2 > \|A\hat{x} - b\|^2$.

We proceed in the same way as for the least-squares problem:

$$\begin{aligned} \|Ax - b\|^2 &= \|(Ax - A\hat{x}) + (A\hat{x} - b)\|^2 \\ &= \|Ax - A\hat{x}\|^2 + \|A\hat{x} - b\|^2 + 2(Ax - A\hat{x})^T(A\hat{x} - b). \end{aligned}$$

Now we expand the last term:

$$\begin{aligned}
2(Ax - A\hat{x})^T(A\hat{x} - b) &= 2(x - \hat{x})^T A^T (A\hat{x} - b) \\
&= -(x - \hat{x})^T C^T z \\
&= -(C(x - \hat{x}))^T z \\
&= 0,
\end{aligned}$$

where we use $2A^T(A\hat{x} - b) = -C^T z$ in the second line and $Cx = C\hat{x} = d$ in the last line. So we have, exactly as in the case of unconstrained least-squares,

$$\|Ax - b\|^2 = \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2.$$

This shows that $\hat{x}$ minimizes $\|Ax - b\|^2$ subject to $Cx = d$. It remains to show that for $x \neq \hat{x}$, $\|A(x - \hat{x})\|^2 > 0$. If this is not the case, then $A(x - \hat{x}) = 0$. We also have $C(x - \hat{x}) = 0$, and so

$$\left[\begin{array}{c} A \\ C \end{array}\right](x - \hat{x}) = 0.$$

By our assumption that the matrix on the left has independent columns, we conclude that $x = \hat{x}$.

**Solving constrained least-squares problems.** We can compute the solution (10.6) of the constrained least-squares problem by forming and solving the KKT equations (10.4).

**Algorithm 10.1** CONSTRAINED LEAST-SQUARES VIA KKT EQUATIONS

**given** an $m \times n$ matrix $A$ and a $p \times n$ matrix $C$ that satisfy (10.5), an $m$-vector $b$, and a $p$-vector $d$.

1. *Form Gram matrix.* Compute $A^T A$.
2. *Solve KKT equations.* Solve KKT equations (10.4) by QR factorization and back substitution.

The second step cannot fail, provided the assumption (10.5) holds. Let us analyze the complexity of this algorithm. The first step is multiplying an $n \times m$ matrix by an $m \times n$ matrix, which requires $2mn^2$ flops. (In fact we can get away with half this number, since the Gram matrix is symmetric, and we only have to compute the entries on and above the diagonal.) The second step requires the solution of a square system of $n + p$ equations, which costs $2(n + p)^3$ flops, so the total is

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

flops. This grows linearly in $m$ and cubicly in $n$ and $p$. The assumption (10.5) implies $p \leq n$, so in terms of order, $(n + p)^3$ can be replaced with $n^3$.

**Solving constrained least-squares problems via QR factorization.** We now give a method for solving the constrained least-squares problem that generalizes the QR factorization method for least-squares problems (algorithm 8.1). We assume that $A$ and $C$ satisfy the conditions (10.5).

We start by rewriting the KKT equations (10.4) as

$$2(A^TA + C^TC)\hat{x} + C^Tw = 2A^Tb, \qquad C\hat{x} = d \tag{10.7}$$

with a new variable $w = z - 2d$. To obtain (10.7) we multiplied the equation $C\hat{x} = d$ on the left by $2C^T$, then added the result to the first equation of (10.4), and replaced the variable $z$ with $w + 2d$.

Next we use the QR factorization

$$\left[\begin{array}{c} A \\ C \end{array}\right] = QR = \left[\begin{array}{c} Q_1 \\ Q_2 \end{array}\right] R \tag{10.8}$$

to simplify (10.7). This factorization exists because the stacked matrix has independent columns, by our assumption (10.5). In (10.8) we also partition $Q$ in two blocks $Q_1$ and $Q_2$, of size $m\times n$ and $p\times n$, respectively. If we make the substitutions $A = Q_1R$, $C = Q_2R$, and $A^TA + C^TC = R^TR$ in (10.7) we obtain

$$2R^TR\hat{x} + R^TQ_2^Tw = 2R^TQ_1^Tb, \qquad Q_2R\hat{x} = d.$$

We multiply the first equation on the left by $R^{-T}$ (which we know exists) to get

$$R\hat{x} = Q_1^Tb - (1/2)Q_2^Tw. \tag{10.9}$$

Substituting this expression into $Q_2R\hat{x} = d$ gives an equation in $w$:

$$Q_2Q_2^Tw = 2Q_2Q_1^Tb - 2d. \tag{10.10}$$

We now use the second part of the assumption (10.5) to show that the matrix $Q_2^T = R^{-T}C^T$ has linearly independent columns. Suppose $Q_2^Tz = R^{-T}C^Tz = 0$. Multiplying with $R^T$ gives $C^Tz = 0$. Since $C$ has linearly independent rows, this implies $z = 0$, and we conclude that the columns of $Q_2^T$ are independent.

The matrix $Q_2^T$ therefore has a QR factorization $Q_2^T = \tilde{Q}\tilde{R}$. Substituting this into (10.10) gives

$$\tilde{R}^T\tilde{R}w = 2\tilde{R}^T\tilde{Q}^TQ_1^Tb - 2d,$$

which we can write as

$$\tilde{R}w = 2\tilde{Q}^TQ_1^Tb - 2\tilde{R}^{-T}d.$$

We can use this to compute $w$, first by computing $\tilde{R}^{-T}d$ (by forward substitution), then forming the right-hand side, and then solving for $w$ using back substitution. Once we know $w$, we can find $\hat{x}$ from (10.9). The method is summarized in the following algorithm.

**Algorithm 10.2** CONSTRAINED LEAST-SQUARES VIA QR FACTORIZATION

**given** an $m \times n$ matrix $A$ and a $p \times n$ matrix $C$ that satisfy (10.5), an $m$-vector $b$, and a $p$-vector $d$.

1. *QR factorizations.* Compute the QR factorizations
$$\begin{bmatrix} A \\ C \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \qquad Q_2^T = \tilde{Q}\tilde{R}.$$
2. Compute $\tilde{R}^{-T}d$ by forward substitution.
3. Form right-hand side and solve
$$\tilde{R}^T w = 2\tilde{Q}^T Q_1^T b - 2R^{-T}d$$
via forward substitution.
4. *Compute $\hat{x}$.* Form right-hand side and solve
$$R\hat{x} = Q_1^T b - (1/2)Q_2^T w$$
by back substitution.

---

In the unconstrained case (when $p = 0$), step 1 reduces to computing the QR factorization of $A$, steps 2 and 3 are not needed, and step 4 reduces to solving $R\hat{x} = Q_1^T b$. This is the same as algorithm 8.1 for solving (unconstrained) least-squares problems.

We now give a complexity analysis. Step 1 involves the QR factorization of an $(m+p) \times n$ and an $n \times p$ matrix, which costs $2(m+p)n^2 + 2np^2$ flops.

Step 2 requires $2p^2$ flops. In step 3, we first evaluate $Q_1^T b$ ($mp$ flops), multiply the result by $Q_2$ ($2pn$ flops), and then solve for $w$ using forward substitution ($2p^2$ flops). Step 4 requires $2mn + 2pn$ flops to form the right-hand side, and $2n^2$ flops to compute $\hat{x}$ via back substitution. The costs of steps 2, 3, and 4 are are quadratic in the dimensions, and so are negligible compared to the cost of step 1, so our final complexity is
$$2(m+p)n^2 + 2np^2$$
flops. The assumption (10.5) implies the inequalities
$$p \leq n \leq m + p,$$
and therefore $(m+p)n^2 \geq np^2$. So the flop count above is no more than $3(m+p)n^2$ flops. In particular, its order is $(m+p)n^2$.

**Solution of least-norm problem.** Here we specialize the solution of the general constrained least-squares problem (10.1) given above to the special case of the least-norm problem (10.2).

We start with the conditions (10.5). The stacked matrix is in this case
$$\begin{bmatrix} I \\ C \end{bmatrix},$$
which always has independent columns. So the conditions (10.5) reduce to: $C$ has independent rows. We make this assumption now.

For the least-norm problem, the KKT equations (10.4) reduce to

$$\begin{bmatrix} 2I & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}.$$

We can solve this using the methods for general constrained least-squares, or derive the solution directly, which we do now. The first block row of this equation is $2\hat{x} + C^T z = 0$, so

$$\hat{x} = -(1/2)C^T z.$$

We substitute this into the second block equation, $C\hat{x} = d$, to obtain

$$-(1/2)CC^T z = d.$$

Since the rows of $C$ are independent, $CC^T$ is invertible, so we have

$$z = -2(CC^T)^{-1}d.$$

Substituting this expression for $z$ into the formula for $\hat{x}$ above gives

$$\hat{x} = C^T(CC^T)^{-1}d. \tag{10.11}$$

We have seen the matrix in this formula before: It is the pseudo-inverse of a wide matrix with independent rows. So we can express the solution of the least-norm problem (10.2) in the very compact form

$$\hat{x} = C^\dagger d.$$

In §7.5, we saw that $C^\dagger$ is a right inverse of $C$; here we see that not only does $\hat{x} = C^\dagger d$ satisfy $Cx = d$, but it gives the vector of least norm that satisfies $Cx = d$.

In §7.5, we also saw that the pseudo-inverse of $C$ can be expressed as $C^\dagger = QR^{-T}$, where $C^T = QR$ is the QR factorization of $C^T$. The solution of the least-norm problem can therefore be expressed as

$$\hat{x} = QR^{-T}d$$

and this leads to an algorithm for solving the least-norm problem via the QR factorization.

**Algorithm 10.3** LEAST-NORM VIA QR FACTORIZATION

```
given a $p \times n$ matrix $C$ with linearly independent rows and a $p$-vector $d$.

    1. QR factorization. Compute the QR factorization $C^T = QR$.
    2. Compute $\hat{x}$. Solve $R^T y = d$ by forward substitution.
    3. Compute $\hat{x} = Qy$.
```

The complexity of this algorithm is dominated by the cost of the QR factorization in step 1, *i.e.*, $2np^2$ flops.

# Appendix A

# Notation

## Vectors

| | |
|---|---|
| $x_i$ | The $i$th element of a vector $x$. |
| $x_{r:s}$ | Subvector with entries from $r$ to $s$. |
| $0$ | Vector with all components zero. |
| $\mathbf{1}$ | Vector with all components one. |
| $e_i$ | The $i$th standard basis vector. |
| $x^T y$ | Inner product of vectors $x$ and $y$. |
| $\|x\|$ | Norm of vector $x$. |
| $\mathbf{rms}(x)$ | RMS value of a vector $x$. |
| $\mathbf{avg}(x)$ | Average of entries of a vector $x$. |
| $\mathbf{std}(x)$ | Standard deviation of a vector $x$. |
| $\mathbf{dist}(x, y)$ | Distance between points $x$ and $y$. |
| $\angle(x, y)$ | Angle between vectors $x$ and $y$. |
| $x \perp y$ | Vectors $x$ and $y$ are orthogonal. |

## Matrices

| | |
|---|---|
| $X_{ij}$ | The $i, j$th element of a matrix $X$. |
| $X_{r:s,p:q}$ | Submatrix with row range $r$ to $s$ and column range $p$ to $q$. |
| $0$ | Matrix with all entries 0. |
| $I$ | Identity matrix. |
| $X^T$ | Transpose of matrix $X$. |
| $X^k$ | Matrix $X$ to the $k$th power. |
| $X^{-1}$ | Inverse of matrix $X$. |
| $X^{-T}$ | Inverse of transpose of matrix $X$. |
| $X^\dagger$ | Pseudo-inverse of matrix $X$. |
| $\mathbf{diag}(x)$ | Diagonal matrix with diagonal entries $x_1, \ldots, x_n$. |

## Functions and derivatives

| | |
|---|---|
| $f : A \to B$ | $f$ is a function on the set $A$ into the set $B$. |
| $\nabla f(z)$ | Gradient of function $f : \mathbf{R}^n \to \mathbf{R}$ at $z$. |
| $Df(z)$ | Derivative (Jacobian) matrix of function $f : \mathbf{R}^n \to \mathbf{R}^m$ at $z$. |

# Appendix B

# Complexity

Here we summarize approximate complexities or flop counts of various operations and algorithms encountered in the book. We drop terms of lower order. When the operands or arguments are sparse, and the operation or algorithm has been modified to take advantage of sparsity, the flops counts can be dramatically lower than those given here.

## Vector operations

In the table below, $x$ and $y$ are $n$-vectors and $a$ is a scalar.

| | |
|---|---|
| $ax$ | $n$ |
| $x+y$ | $n$ |
| $x^Ty$ | $2n$ |
| $\|x\|$ | $2n$ |
| $\|x-y\|$ | $3n$ |
| $\mathbf{rms}(x)$ | $2n$ |
| $\mathbf{std}(x)$ | $4n$ |
| $\angle(x,y)$ | $6n$ |

## Matrix operations

In the table below, $A$ and $B$ are $m \times n$ matrices, $C$ is an $m \times p$ matrix, $x$ is an $n$-vector, and $a$ is a scalar.

| | |
|---|---|
| $aA$ | $mn$ |
| $A+B$ | $mn$ |
| $Ax$ | $2mn$ |
| $AC$ | $2mnp$ |
| $A^TA$ | $mn^2$ |

## Factorization and inverses

In the table below, $A$ is a tall or square $m \times n$ matrix, $R$ is an $n \times n$ triangular matrix, and $b$ is an $n$-vector. We assume the factorization or inverses exist; in particular in any expression involving $A^{-1}$, $A$ must be square.

| | |
|---|---|
| QR factorization of $A$ | $2mn^2$ |
| $R^{-1}b$ | $n^2$ |
| $A^{-1}b$ | $2n^3$ |
| $A^{-1}$ | $3n^3$ |
| $A^\dagger$ | $2mn^2$ |

## Solving least-squares problems

In the table below, $A$ is an $m \times n$ matrix, $C$ is a wide $p \times n$ matrix, and $b$ is an $m$-vector. We assume the associated independence conditions hold.

| | |
|---|---|
| minimize $\|Ax - b\|^2$ | $2mn^2$ |
| minimize $\|Ax - b\|^2$ subject to $Cx = d$ | $2(m+p)n^2$ |
| minimize $\|x\|^2$ subject to $Cx = d$ | $2np^2$ |