costs $2n^2$ flops. The total cost is $3n^2$ flops. On the other hand if we first evaluate the inner product $b^T c$, the cost is $2n$ flops, and we only need to store one number (the result). Multiplying the vector a by this number costs n flops, so the total cost is $3n$ flops. For n large, there is a dramatic difference between $3n$ and $3n^2$ flops. (The storage requirements are also dramatically different for the two methods of evaluating $ab^T c$: one number versus n^2 numbers.)

10.2 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 of matrix multiplication as composition of linear functions, 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 2×2 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.

Second difference matrix. As a more interesting example of composition of linear functions, consider the $(n-1) \times n$ difference matrix D_n defined in (6.5). (We use the subscript n here to denote size of D.) Let D_{n-1} denote the $(n-2) \times (n-1)$ difference matrix. Their product $D_{n-1}D_n$ is called the *second difference matrix*, and sometimes denoted Δ.

We can interpret Δ in terms of composition of linear functions. Multiplying an n-vector x by D_n yields the $(n-1)$-vector of consecutive differences of the entries:

$$D_n x = (x_2 - x_1, \ldots, x_n - x_{n-1}).$$

Multiplying this vector by D_{n-1} gives the $(n-2)$-vector of consecutive differences of consecutive differences (or second differences) of x:

$$D_{n-1}D_n x = (x_1 - 2x_2 + x_3,\ x_2 - 2x_3 + x_4,\ \ldots,\ x_{n-2} - 2x_{n-1} + x_n).$$

The $(n-2) \times n$ product matrix $\Delta = D_{n-1}D_n$ is the matrix associated with the second difference function.

For the case $n = 5$, $\Delta = D_{n-1}D_n$ has the form

$$\begin{bmatrix} 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

The left-hand matrix Δ is associated with the second difference linear function that maps 5-vectors into 3-vectors. The middle matrix D_4 is associated with the difference function that maps 4-vectors into 3-vectors. The right-hand matrix D_5 is associated with the difference function that maps 5-vectors into 4-vectors.

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:

$$\frac{\partial h_i}{\partial x_j}(z) = \frac{\partial f_i}{\partial y_1}(g(z))\frac{\partial g_1}{\partial x_j}(z) + \cdots + \frac{\partial f_i}{\partial y_p}(g(z))\frac{\partial g_p}{\partial x_j}(z)$$

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 f is 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 $\nabla h(z) = A^T \nabla f(Ax + b)$.

Linear dynamical system with state feedback. We consider a time-invariant linear dynamical system with n-vector state x_t and m-vector input u_t, with dynamics

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

Here we think of the input u_t as something we can manipulate, *e.g.*, the control surface deflections for an airplane or the amount of material we order or move in a supply chain. In *state feedback control* the state x_t is measured, and the input u_t is a linear function of the state, expressed as

$$u_t = Kx_t,$$

where K is the $m \times n$ *state-feedback gain matrix*. The term *feedback* refers to the idea that the state is measured, and then (after multiplying by K) fed back into the system, via the input. This leads to a loop, where the state affects the input, and the input affects the (next) state. State feedback is very widely used in many applications. (In §17.2.3 we will see methods for choosing or designing an appropriate state feedback matrix.)

With state feedback, we have

$$x_{t+1} = Ax_t + Bu_t = Ax_t + B(Kx_t) = (A + BK)x_t, \quad t = 1, 2, \ldots.$$

This recursion is called the *closed-loop system*. The matrix $A + BK$ is called the *closed-loop dynamics matrix*. (In this context, the recursion $x_{t+1} = Ax_t$ is called the *open-loop system*. It gives the dynamics when $u_t = 0$.)

10.3 Matrix power

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 all nonnegative integer values of k and l.

We should mention one ambiguity in matrix power notation that occasionally arises. When A is a square matrix and T is a nonnegative integer, A^T can mean either the transpose of the matrix A or its Tth power. Usually which is meant is clear from the context, or the author explicitly states which meaning is intended. To avoid this ambiguity, some authors use a different symbol for the transpose, such as A^{T} (with the superscript in roman font) or A', or avoid referring to the Tth power of a matrix. When A is not square there is no ambiguity, since A^T can only be the transpose in this case.

Other matrix powers. Matrix powers A^k with k a negative integer will be discussed in §11.2. Non-integer powers, such as $A^{1/2}$ (the matrix squareroot), need not make sense, or can be ambiguous, unless certain conditions on A hold. This is an advanced topic in linear algebra that we will not pursue in this book.

Paths in a directed graph. Suppose A is the $n \times n$ adjacency matrix of a directed graph with n vertices:

$$A_{ij} = \begin{cases} 1 & \text{there is a edge from vertex } j \text{ to vertex } i \\ 0 & \text{otherwise} \end{cases}$$

(see page 112). A *path* of length ℓ is a sequence of $\ell + 1$ vertices, with an edge from the first to the second vertex, an edge from the second to third vertex, and so on. We say the path goes from the first vertex to the last one. An edge can be considered a path of length one. By convention, every vertex has a path of length zero (from the vertex to itself).

The elements of the matrix powers A^ℓ have a simple meaning in terms of paths in the graph. First examine the expression for the i, j element of the square of A:

$$(A^2)_{ij} = \sum_{k=1}^{n} A_{ik}A_{kj}.$$