CHAPTER 4

Designing and Analyzing Linear Systems

CONTENTS

NOW that we can solve linear systems of equations, we will show how to apply this machinery to several practical problems. The algorithms introduced in the previous chapter can be applied directly to produce the desired output in each case.

While LU factorization and Gaussian elimination are guaranteed to solve each of these problems in polynomial time, a natural question is whether there exist more efficient or stable algorithms if we know more about the structure of a particular linear system. Thus, we will examine the matrices constructed in the initial examples to reveal special properties that some of them have in common. Designing algorithms specifically for these classes of matrices will provide speed and numerical advantages, at the cost of generality.

Finally, we will return to concepts from Chapter 2 to design heuristics evaluating how much we can trust the solution $\vec{x}$ to a linear system $A\vec{x} = \vec{b}$, in the presence of rounding and other sources of error. This aspect of analyzing linear systems must be considered when designing reliable and consistent implementations of numerical algorithms.

4.1 SOLUTION OF SQUARE SYSTEMS

In the previous chapter, we only considered square, invertible matrices A when solving $A\vec{x} = \vec{b}$. While this restriction does preclude some important cases, many if not most

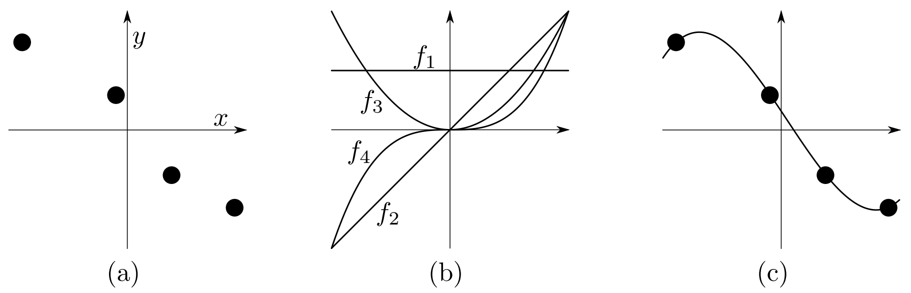


Figure 4.1 (a) The input for regression, a set of $(x^{(k)}, y^{(k)})$ pairs; (b) a set of basis functions $\{f_1, f_2, f_3, f_4\}$; (c) the output of regression, a set of coefficients $c_1, \ldots, c_4$ such that the linear combination $\sum_{k=1}^{4} c_k f_k(x)$ goes through the data points.

applications of linear systems can be posed in terms of square, invertible matrices. We explore a few of these applications below.

4.1.1 Regression

We start with an application from data analysis known as *regression*. Suppose we carry out a scientific experiment and wish to understand the structure of the experimental results. One way to model these results is to write the *independent variables* of a given trial in a vector $\vec{x} \in \mathbb{R}^n$ and to think of the *dependent variable* as a function $f(\vec{x}) : \mathbb{R}^n \to \mathbb{R}$. Given a few $(\vec{x}, f(\vec{x}))$ pairs, our goal is to predict the output of $f(\vec{x})$ for a new $\vec{x}$ without carrying out the full experiment.

Example 4.1 (Biological experiment)**.** Suppose we wish to measure the effects of fertilizer, sunlight, and water on plant growth. We could do a number of experiments applying different amounts of fertilizer (in cm^3), sunlight (in watts), and water (in ml) to a plant and measuring the height of the plant after a few days. Assuming plant height is a direct function of these variables, we can model our observations as samples from a function $f : \mathbb{R}^3 \to \mathbb{R}$ that takes the three parameters we wish to test and outputs the height of the plant at the end of the experimental trial.

In *parametric* regression, we additionally assume that we know the structure of f ahead of time. For example, suppose we assume that f is linear:

$$f(\vec{x}) = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n.$$

Then, our goal becomes more concrete: to estimate the coefficients $a_1, \ldots, a_n$.

We can carry out n experiments to reveal $y^{(k)} \equiv f(\vec{x}^{(k)})$ for samples $\vec{x}^{(k)}$, where $k \in \{1, \ldots, n\}$. For the linear example, plugging into the formula for f shows a set of statements:

$$\begin{aligned}
y^{(1)} = f(\vec{x}^{(1)}) &= a_1 x_1^{(1)} + a_2 x_2^{(1)} + \cdots + a_n x_n^{(1)} \\
y^{(2)} = f(\vec{x}^{(2)}) &= a_1 x_1^{(2)} + a_2 x_2^{(2)} + \cdots + a_n x_n^{(2)} \\
&\vdots
\end{aligned}$$

Contrary to our earlier notation $A\vec{x} = \vec{b}$, the unknowns here are the a_i's, *not* the $\vec{x}^{(k)}$'s. With this notational difference in mind, if we make exactly n observations we can write

$$\begin{pmatrix} - & \vec{x}^{(1)\top} & - \\ - & \vec{x}^{(2)\top} & - \\ & \vdots & \\ - & \vec{x}^{(n)\top} & - \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix}.$$

In other words, if we carry out n trials of our experiment and write the independent variables in the columns of a matrix $X \in \mathbb{R}^{n\times n}$ and the dependent variables in a vector $\vec{y} \in \mathbb{R}^n$, then the coefficients $\vec{a}$ can be recovered by solving the linear system $X^\top \vec{a} = \vec{y}$.

We can generalize this method to certain nonlinear forms for the function f using an approach illustrated in Figure 4.1. The key is to write f as a linear combination of *basis functions*. Suppose $f(\vec{x})$ takes the form

$$f(\vec{x}) = a_1 f_1(\vec{x}) + a_2 f_2(\vec{x}) + \cdots + a_m f_m(\vec{x}),$$

where $f_k : \mathbb{R}^n \to \mathbb{R}$ and we wish to estimate the parameters a_k. Then, by a parallel derivation given m observations of the form $\vec{x}^{(k)} \mapsto y^{(k)}$ we can find the parameters by solving:

$$\begin{pmatrix} f_1(\vec{x}^{(1)}) & f_2(\vec{x}^{(1)}) & \cdots & f_m(\vec{x}^{(1)}) \\ f_1(\vec{x}^{(2)}) & f_2(\vec{x}^{(2)}) & \cdots & f_m(\vec{x}^{(2)}) \\ \vdots & \vdots & \cdots & \vdots \\ f_1(\vec{x}^{(m)}) & f_2(\vec{x}^{(m)}) & \cdots & f_m(\vec{x}^{(m)}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}.$$

That is, even if the f's are nonlinear, we can learn weights a_k using purely linear techniques.

Example 4.2 (Linear regression)**.** The system $X^\top \vec{a} = \vec{y}$ from our initial example can be recovered from the general formulation by taking $f_k(\vec{x}) = x_k$.

Example 4.3 (Polynomial regression)**.** As in Figure 4.1, suppose that we observe a function of a single variable $f(x)$ and wish to write it as an $(n-1)$-st degree polynomial

$$f(x) \equiv a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}.$$

Given n pairs $x^{(k)} \mapsto y^{(k)}$, we can solve for the parameters $\vec{a}$ via the system

$$\begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \cdots & (x^{(1)})^{n-1} \\ 1 & x^{(2)} & (x^{(2)})^2 & \cdots & (x^{(2)})^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x^{(n)} & (x^{(n)})^2 & \cdots & (x^{(n)})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix}.$$

In other words, we take $f_k(x) = x^{k-1}$ in the general form above. Incidentally, the matrix on the left-hand side of this relationship is known as a Vandermonde matrix.

As an example, suppose we wish to find a parabola $y = ax^2 + bx + c$ going through $(-1, 1)$, $(0, -1)$, and $(2, 7)$. We can write the Vandermonde system in two ways:

$$\left\{ \begin{array}{rl} a(-1)^2 + b(-1) + c & = 1 \\ a(0)^2 + b(0) + c & = -1 \\ a(2)^2 + b(2) + c & = 7 \end{array} \right\} \Longleftrightarrow \begin{pmatrix} 1 & -1 & (-1)^2 \\ 1 & 0 & 0^2 \\ 1 & 2 & 2^2 \end{pmatrix} \begin{pmatrix} c \\ b \\ a \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 7 \end{pmatrix}.$$

Gaussian elimination on this system shows $(a, b, c) = (2, 0, -1)$, corresponding to the polynomial $y = 2x^2 - 1$.

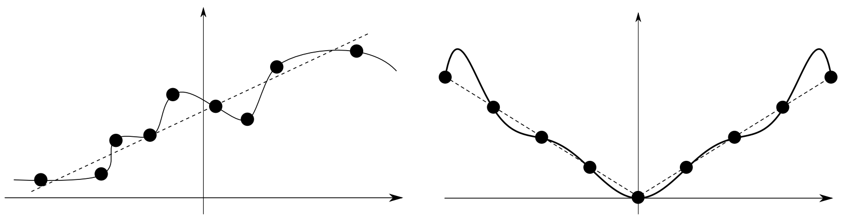


Figure 4.2 Drawbacks of fitting function values exactly: (a) noisy data might be better represented by a simple function rather than a complex curve that touches every data point and (b) the basis functions might not be tuned to the function being sampled. In (b), we fit a polynomial of degree eight to nine samples from $f(x) = |x|$ but would have been more successful using a basis of line segments.

Example 4.4 (Oscillation)**.** A foundational notion from signal processing for audio and images is the decomposition of a function into a linear combination of cosine or sine waves at different frequencies. This decomposition of a function defines its *Fourier transform.*

As the simplest possible case, we can try to recover the parameters of a single-frequency wave. Suppose we wish to find parameters a and ϕ of a function $f(x) = a\cos(x + \phi)$ given two (x, y) samples satisfying $y^{(1)} = f(x^{(1)})$ and $y^{(2)} = f(x^{(2)})$. Although this setup as we have written it is nonlinear, we can recover a and ϕ using a linear system after some mathematical transformations.

From trigonometry, any function of the form $g(x) = a_1 \cos x + a_2 \sin x$ can be written $g(x) = a\cos(x + \phi)$ after applying the formulae

$$a = \sqrt{a_1^2 + a_2^2} \qquad\qquad \phi = -\arctan\frac{a_2}{a_1}.$$

We can find $f(x)$ by applying the linear method to compute the coefficients a_1 and a_2 in $g(x)$ and then using these formulas to find a and ϕ. This construction can be extended to fitting functions of the form $f(x) = \sum_k a_k \cos(x + \phi_k)$, giving one way to motivate the discrete Fourier transform of f, explored in Exercise 4.15.

4.1.2 Least-Squares

The techniques in §4.1.1 provide valuable methods for finding a continuous f matching a set of data pairs $\vec{x}_k \mapsto y_k$ *exactly*. For this reason, they are called *interpolation* schemes, which we will explore in detail in Chapter 13. They have two related drawbacks, illustrated in Figure 4.2:

- There might be some error in measuring the values $\vec{x}_k$ and y_k. In this case, a simpler $f(\vec{x})$ satisfying the approximate relationship $f(\vec{x}_k) \approx y_k$ may be acceptable or even preferable to an exact $f(\vec{x}_k) = y_k$ that goes through each data point.
- If there are m functions $f_1, \ldots, f_m$, then we use exactly m observations $\vec{x}_k \mapsto y_k$. Additional observations have to be thrown out, or we have to introduce more f_k's, which can make the resulting function $f(\vec{x})$ increasingly complicated.

Both of these issues are related to the larger problem of *over-fitting*: Fitting a function with n degrees of freedom to n data points leaves no room for measurement error.

More broadly, suppose we wish to solve the linear system $A\vec{x} = \vec{b}$ for $\vec{x}$. If we denote row k of A as $\vec{r}_k^\top$, then the system looks like

$$\begin{aligned}
\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} &= \begin{pmatrix} - & \vec{r}_1^\top & - \\ - & \vec{r}_2^\top & - \\ \vdots & \vdots & \vdots \\ - & \vec{r}_n^\top & - \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \text{ by expanding } A\vec{x} \\
&= \begin{pmatrix} \vec{r}_1 \cdot \vec{x} \\ \vec{r}_2 \cdot \vec{x} \\ \vdots \\ \vec{r}_n \cdot \vec{x} \end{pmatrix} \text{ by definition of matrix multiplication.}
\end{aligned}$$

From this perspective, each row of the system corresponds to a separate observation of the form $\vec{r}_k \cdot \vec{x} = b_k$. That is, an alternative way to interpret the linear system $A\vec{x} = \vec{b}$ is that it encodes n statements of the form, "The dot product of $\vec{x}$ with $\vec{r}_k$ is b_k."

A tall system $A\vec{x} = \vec{b}$ where $A \in \mathbb{R}^{m\times n}$ and $m > n$ encodes more than n of these dot product observations. When we make more than n observations, however, they may be *incompatible*; as explained §3.1, tall systems do not have to admit a solution.

When we cannot solve $A\vec{x} = \vec{b}$ exactly, we can relax the problem and try to find an approximate solution $\vec{x}$ satisfying $A\vec{x} \approx \vec{b}$. One of the most common ways to solve this problem, known as *least-squares*, is to ask that the residual $\vec{b} - A\vec{x}$ be as small as possible by minimizing the norm $\|\vec{b} - A\vec{x}\|_2$. If there is an exact solution $\vec{x}$ satisfying the tall system $A\vec{x} = \vec{b}$, then the minimum of this energy is zero, since norms are nonnegative and in this case $\|\vec{b} - A\vec{x}\|_2 = \|\vec{b} - \vec{b}\|_2 = 0$.

Minimizing $\|\vec{b} - A\vec{x}\|_2$ is the same as minimizing $\|\vec{b} - A\vec{x}\|_2^2$, which we expanded in Example 1.16 to:

$$\|\vec{b} - A\vec{x}\|_2^2 = \vec{x}^\top A^\top A\vec{x} - 2\vec{b}^\top A\vec{x} + \|\vec{b}\|_2^2.^*$$

The gradient of this expression with respect to $\vec{x}$ must be zero at its minimum, yielding the following system:

$$\begin{aligned}
\vec{0} &= 2A^\top A\vec{x} - 2A^\top\vec{b}, \\
\text{or equivalently,} \quad A^\top A\vec{x} &= A^\top\vec{b}.
\end{aligned}$$

This famous relationship is worthy of a theorem:

Theorem 4.1 (Normal equations)**.** Minima of the residual norm $\|\vec{b} - A\vec{x}\|_2$ for $A \in \mathbb{R}^{m\times n}$ (with no restriction on m or n) satisfy $A^\top A\vec{x} = A^\top\vec{b}$.

The matrix $A^\top A$ is sometimes called a *Gram matrix*. If at least n rows of A are linearly independent, then $A^\top A \in \mathbb{R}^{n\times n}$ is invertible. In this case, the minimum residual occurs uniquely at $(A^\top A)^{-1}A^\top\vec{b}$. Put another way:

In the overdetermined case, solving the least-squares problem $A\vec{x} \approx \vec{b}$ is equivalent to solving the *square* system $A^\top A\vec{x} = A^\top\vec{b}$.

Via the normal equations, we can solve tall systems with $A \in \mathbb{R}^{m\times n}$, $m \geq n$, using algorithms for square matrices.

*If this result is not familiar, it may be valuable to return to the material in §1.4 at this point for review.

4.1.3 Tikhonov Regularization

When solving linear systems, the underdetermined case $m < n$ is considerably more difficult to handle due to increased ambiguity. As discussed in §3.1, in this case we lose the possibility of a *unique* solution to $A\vec{x} = \vec{b}$. To choose between the possible solutions, we must make an additional assumption on $\vec{x}$ to obtain a unique solution, e.g., that it has a small norm or that it contains many zeros. Each such *regularizing* assumption leads to a different solution algorithm. The particular choice of a regularizer may be application-dependent, but here we outline a general approach commonly applied in statistics and machine learning; we will introduce an alternative in §7.2.1 after introducing the singular value decomposition (SVD) of a matrix.

When there are multiple vectors $\vec{x}$ that minimize $\|A\vec{x} - \vec{b}\|_2^2$, the least-squares energy function is *insufficient* to isolate a single output. For this reason, for fixed $\alpha > 0$, we might introduce an additional term to the minimization problem:

$$\min_{\vec{x}} \|A\vec{x} - \vec{b}\|_2^2 + \alpha\|\vec{x}\|_2^2.$$

This second term is known as a *Tikhonov regularizer*. When $0 < \alpha \ll 1$, this optimization effectively asks that among the minimizers of $\|A\vec{x} - \vec{b}\|_2$ we would prefer those with small norm $\|\vec{x}\|_2$; as α increases, we prioritize the norm of $\vec{x}$ more. This energy is the product of an "Occam's razor" philosophy: In the absence of more information about $\vec{x}$, we might as well choose an $\vec{x}$ with small entries.

To minimize this new objective, we take the derivative with respect to $\vec{x}$ and set it equal to zero:

$$\vec{0} = 2A^\top A\vec{x} - 2A^\top\vec{b} + 2\alpha\vec{x},$$

or equivalently

$$(A^\top A + \alpha I_{n\times n})\vec{x} = A^\top\vec{b}.$$

So, if we wish to introduce Tikhonov regularization to a linear problem, all we have to do is add α down the diagonal of the Gram matrix $A^\top A$.

When $A\vec{x} = \vec{b}$ is underdetermined, the matrix $A^\top A$ is not invertible. The new Tikhonov term resolves this issue, since for $\vec{x} \neq \vec{0}$,

$$\vec{x}^\top(A^\top A + \alpha I_{n\times n})\vec{x} = \|A\vec{x}\|_2^2 + \alpha\|\vec{x}\|_2^2 > 0.$$

The strict $>$ holds because $\vec{x} \neq \vec{0}$; it implies that $A^\top A + \alpha I_{n\times n}$ cannot have a null space vector $\vec{x}$. Hence, regardless of A, the Tikhonov-regularized system of equations is invertible. In the language we will introduce in §4.2.1, it is *positive definite*.

Tikhonov regularization is effective for dealing with null spaces and numerical issues. When A is poorly conditioned, adding this type of regularization can improve conditioning even when the original system was solvable. We acknowledge two drawbacks, however, that can require more advanced algorithms when they are relevant:

- The solution $\vec{x}$ of the Tikhonov-regularized system no longer satisfies $A\vec{x} = \vec{b}$ exactly.
- When α is small, the matrix $A^\top A + \alpha I_{n\times n}$ is invertible but may be poorly conditioned. Increasing α solves this problem at the cost of less accurate solutions to $A\vec{x} = \vec{b}$.

When the columns of A span $\mathbb{R}^m$, an alternative to Tikhonov regularization is to minimize $\|\vec{x}\|_2$ with the "hard" constraint $A\vec{x} = \vec{b}$. Exercise 4.7 shows that this least-norm solution is given by $\vec{x} = A^\top(AA^\top)^{-1}\vec{b}$, a similar formula to the normal equations for least-squares.

Example 4.5 (Tikhonov regularization)**.** Suppose we pose the following linear system:

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.00001 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 0.99 \end{pmatrix}.$$

This system is solved by $\vec{x} = (1001, -1000)$.

The scale of this $\vec{x} \in \mathbb{R}^2$, however, is much larger than that of any values in the original problem. We can use Tikhonov regularization to encourage smaller values in $\vec{x}$ that still solve the linear system approximately. In this case, the Tikhonov system is

$$\left[\begin{pmatrix} 1 & 1 \\ 1 & 1.00001 \end{pmatrix}^\top \begin{pmatrix} 1 & 1 \\ 1 & 1.00001 \end{pmatrix} + \alpha I_{2\times 2}\right] \vec{x} = \begin{pmatrix} 1 & 1 \\ 1 & 1.00001 \end{pmatrix}^\top \begin{pmatrix} 1 \\ 0.99 \end{pmatrix},$$

or equivalently,

$$\begin{pmatrix} 2+\alpha & 2.00001 \\ 2.00001 & 2.0000200001 + \alpha \end{pmatrix} \vec{x} = \begin{pmatrix} 1.99 \\ 1.9900099 \end{pmatrix}.$$

As α increases, the regularizer becomes stronger. Some example solutions computed numerically are below:

$$\begin{aligned} \alpha = 0.00001 &\longrightarrow \vec{x} \approx (0.499998, 0.494998) \\ \alpha = 0.001 &\longrightarrow \vec{x} \approx (0.497398, 0.497351) \\ \alpha = 0.1 &\longrightarrow \vec{x} \approx (0.485364, 0.485366). \end{aligned}$$

Even with a tiny amount of regularization, these solutions approximate the symmetric near-solution $\vec{x} \approx (0.5, 0.5)$, which has much smaller magnitude. If α becomes *too* large, regularization overtakes the system and $\vec{x} \to (0, 0)$.

4.1.4 Image Alignment

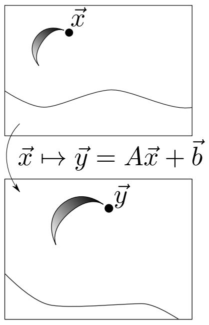
Suppose we take two photographs of the same scene from different positions. One common task in computer vision and graphics is to stitch them together to make a single larger image. To do so, the user (or an automatic system) marks p pairs of points $\vec{x}_k, \vec{y}_k \in \mathbb{R}^2$ such that for each k the location $\vec{x}_k$ in image one corresponds to the location $\vec{y}_k$ in image two. Then, the software automatically warps the second image onto the first or vice versa such that the pairs of points are aligned.

When the camera makes a small motion, a reasonable assumption is that there exists some transformation matrix $A \in \mathbb{R}^{2\times 2}$ and a translation vector $\vec{b} \in \mathbb{R}^2$ such that for all k,

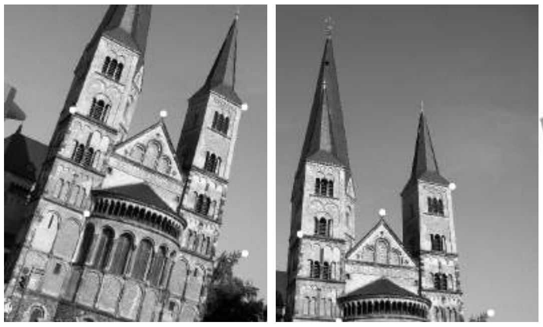
$$\vec{y}_k \approx A\vec{x}_k + \vec{b}.$$

That is, position $\vec{x}$ on image one should correspond to position $A\vec{x} + \vec{b}$ on image two. Figure 4.3(a) illustrates this notation. With this assumption, given a set of corresponding pairs $(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_p, \vec{y}_p)$, our goal is to compute the A and $\vec{b}$ matching these points as closely as possible.

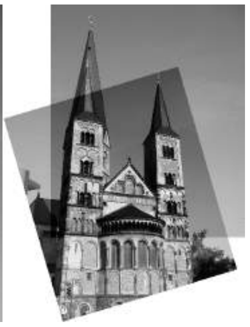
Beyond numerical issues, mistakes may have been made while locating the corresponding points, and we must account for approximation error due to the slightly nonlinear camera projection of real-world lenses. To address this potential for misalignment, rather than



(a)



(b) Input images with keypoints



(c) Aligned images

Figure 4.3 (a) The image alignment problem attempts to find the parameters A and $\vec{b}$ of a transformation from one image of a scene to another using labeled keypoints $\vec{x}$ on the first image paired with points $\vec{y}$ on the second. As an example, keypoints marked in white on the two images in (b) are used to create (c) the aligned image.

requiring that the marked points match exactly, we can ask that they are matched in a least-squares sense. To do so, we solve the following minimization problem:

$$\min_{A,\vec{b}} \sum_{k=1}^{p} \|(A\vec{x}_k + \vec{b}) - \vec{y}_k\|_2^2.$$

This problem has six unknowns total, the four elements of A and the two elements of $\vec{b}$. Figure 4.3(b,c) shows typical output for this method; five keypoints rather than the required three are used to stabilize the output transformation using least-squares.

This objective is a sum of squared linear expressions in the unknowns A and $\vec{b}$, and we will show that it can be minimized using a linear system. Define

$$f(A,\vec{b}) \equiv \sum_k \|(A\vec{x}_k + \vec{b}) - \vec{y}_k\|_2^2.$$

We can simplify f as follows:

$$\begin{aligned} f(A,\vec{b}) &= \sum_k (A\vec{x}_k + \vec{b} - \vec{y}_k)^\top (A\vec{x}_k + \vec{b} - \vec{y}_k) \text{ since } \|\vec{v}\|_2^2 = \vec{v}^\top\vec{v} \\ &= \sum_k \left[\vec{x}_k^\top A^\top A\vec{x}_k + 2\vec{x}_k^\top A^\top \vec{b} - 2\vec{x}_k^\top A^\top \vec{y}_k + \vec{b}^\top\vec{b} - 2\vec{b}^\top\vec{y}_k + \vec{y}_k^\top\vec{y}_k\right] \\ &\qquad \text{where terms with leading 2 apply the fact } \vec{a}^\top\vec{b} = \vec{b}^\top\vec{a}. \end{aligned}$$

To find where f is minimized, we differentiate it with respect to $\vec{b}$ and with respect to the elements of A, and set these derivatives equal to zero. This leads to the following system:

$$\begin{aligned} 0 = \nabla_{\vec{b}} f(A,\vec{b}) &= \sum_k \left[2A\vec{x}_k + 2\vec{b} - 2\vec{y}_k\right] \\ 0 = \nabla_A f(A,\vec{b}) &= \sum_k \left[2A\vec{x}_k\vec{x}_k^\top + 2\vec{b}\vec{x}_k^\top - 2\vec{y}_k\vec{x}_k^\top\right] \text{ by the identities in Exercise 4.3.} \end{aligned}$$

In the second equation, we use the gradient $\nabla_A f$ to denote the *matrix* whose entries are $(\nabla_A f)_{ij} \equiv \partial f/\partial A_{ij}$. Simplifying somewhat, if we define $X \equiv \sum_k \vec{x}_k\vec{x}_k^\top$, $\vec{x}_{\text{sum}} \equiv \sum_k \vec{x}_k$,

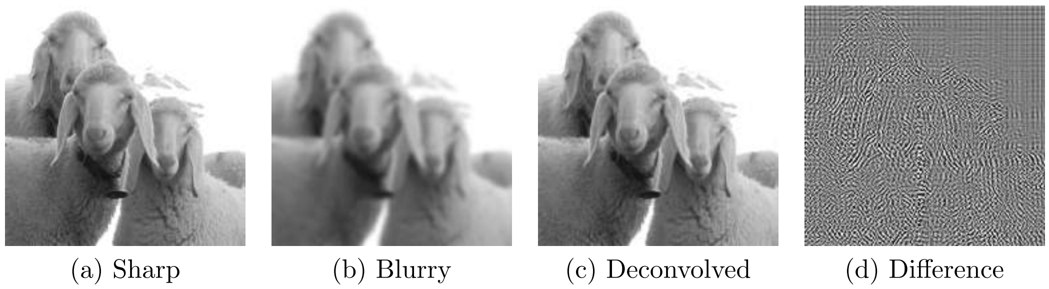


Figure 4.4 Suppose rather than taking (a) the sharp image, we accidentally take (b) a blurry photo; then, deconvolution can be used to recover (c) a sharp approximation of the original image. The difference between (a) and (c) is shown in (d); only high-frequency detail is different between the two images.

$\vec{y}_{\text{sum}} \equiv \sum_k \vec{y}_k$, and $C \equiv \sum_k \vec{y}_k \vec{x}_k^\top$, then the optimal A and $\vec{b}$ satisfy the following linear system of equations:

$$\begin{aligned} A\vec{x}_{\text{sum}} + p\vec{b} &= \vec{y}_{\text{sum}} \\ AX + \vec{b}\vec{x}_{\text{sum}}^\top &= C. \end{aligned}$$

This system is linear in the unknowns A and $\vec{b}$; Exercise 4.4 expands it explicitly using a 6×6 matrix.

This example illustrates a larger pattern in modeling using least-squares. We started by defining a desirable relationship between the unknowns, namely $(A\vec{x} + \vec{b}) - \vec{y} \approx \vec{0}$. Given a number of data points $(\vec{x}_k, \vec{y}_k)$, we designed an objective function f measuring the quality of potential values for the unknowns A and $\vec{b}$ by summing up the squared norms of expressions we wished to equal zero: $\sum_k \|(A\vec{x}_k + \vec{b}) - \vec{y}_k\|_2^2$. Differentiating this sum gave a *linear* system of equations to solve for the best possible choice. This pattern is a common source of optimization problems that can be solved linearly and essentially is a subtle application of the normal equations.

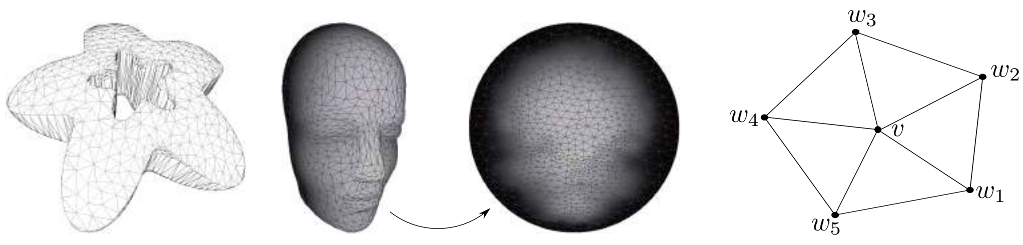
4.1.5 Deconvolution

An artist hastily taking pictures of a scene may accidentally take photographs that are slightly out of focus. While a photo that is completely blurred may be a lost cause, if there is only localized or small-scale blurring, we may be able to recover a sharper image using computational techniques. One strategy is *deconvolution*, explained below; an example test case of the method outlined below is shown in Figure 4.4.

We can think of a grayscale photograph as a point in $\mathbb{R}^p$, where p is the number of pixels it contains; each pixel's intensity is stored in a different dimension. If the photo is in color, we may need red, green, and blue intensities per pixel, yielding a similar representation in $\mathbb{R}^{3p}$. Regardless, most image blurs are linear, including Gaussian convolution or operations averaging a pixel's intensity with those of its neighbors. In image processing, these operators can be encoded using a matrix G taking a sharp image $\vec{x}$ to its blurred counterpart $G\vec{x}$.

Suppose we take a blurry photo $\vec{x}_0 \in \mathbb{R}^p$. Then, we could try to recover the underlying sharp image $\vec{x} \in \mathbb{R}^p$ by solving the least-squares problem

$$\min_{\vec{x} \in \mathbb{R}^p} \|\vec{x}_0 - G\vec{x}\|_2^2.$$



(a) Triangle mesh (b) Parameterization (c) Harmonic condition

Figure 4.5 (a) An example of a triangle mesh, the typical structure used to represent three-dimensional shapes in computer graphics. (b) In mesh parameterization, we seek a map from a three-dimensional mesh (left) to the two-dimensional image plane (right); the right-hand side shown here was computed using the method suggested in §4.1.6. (c) The harmonic condition is that the position of vertex v is the average of the positions of its neighbors $w_1, \ldots, w_5$.

This model assumes that when you blur $\vec{x}$ with G, you get the observed photo $\vec{x}_0$. By the same construction as previous sections, if we know G, then this problem can be solved using linear methods.

In practice, this optimization might be unstable since it is solving a difficult inverse problem. In particular, many pairs of distinct images look very similar after they are blurred, making the reverse operation challenging. One way to stabilize the output of deconvolution is to use Tikhonov regularization, from §4.1.3:

$$\min_{\vec{x}\in\mathbb{R}^p} \|\vec{x}_0 - G\vec{x}\|_2^2 + \alpha\|\vec{x}\|_2^2.$$

More complex versions may constrain $\vec{x} \geq 0$, since negative intensities are not reasonable, but adding such a constraint makes the optimization nonlinear and better solved by the methods we will introduce starting in Chapter 10.

4.1.6 Harmonic Parameterization

Systems for animation often represent geometric objects in a scene using *triangle meshes*, sets of points linked together into triangles as in Figure 4.5(a). To give these meshes fine textures and visual detail, a common practice is to store a detailed color texture as an image or photograph, and to map this texture onto the geometry. Each vertex of the mesh then carries not only its geometric location in space but also *texture coordinates* representing its position on the texture plane.

Mathematically, a mesh can be represented as a collection of n vertices $V \equiv \{v_1, \ldots, v_n\}$ linked in pairs by edges $E \subseteq V \times V$. Geometrically, each vertex $v \in V$ is associated with a location $\vec{x}(v)$ in three-dimensional space $\mathbb{R}^3$. Additionally, we will decorate each vertex with a texture coordinate $\vec{t}(v) \in \mathbb{R}^2$ describing its location in the image plane. It is desirable for these positions to be laid out smoothly to avoid squeezing or stretching the texture relative to the geometry of the surface. With this criterion in mind, the problem of *parameterization* is to fill in the positions $\vec{t}(v)$ for all the vertices $v \in V$ given a few positions laid out manually; desirable mesh parameterizations minimize the geometric distortion of the mesh from its configuration in three-dimensional space to the plane. Surprisingly, many state-of-the-art parameterization algorithms involve little more than a linear solve; we will outline one method originally proposed in [123].

For simplicity, suppose that the mesh has disk topology, meaning that it can be mapped to the interior of a circle in the plane, and that we have fixed the location of each vertex on its boundary $B \subseteq V$. The job of the parameterization algorithm then is to fill in positions for the interior vertices of the mesh. This setup and the output of the algorithm outlined below are shown in Figure 4.5(b).

For a vertex $v \in V$, take $N(v)$ to be the set of neighbors of v on the mesh, given by

$$N(v) \equiv \{w \in V : (v, w) \in E\}.$$

Then, for each vertex $v \in V \backslash B$, a reasonable criterion for parameterization quality is that v should be located at the center of its neighbors, illustrated in Figure 4.5(c). Mathematically, this condition is written

$$\vec{t}(v) = \frac{1}{|N(v)|} \sum_{w \in N(v)} \vec{t}(w).$$

Using this expression, we can associate each $v \in V$ with a linear condition either fixing its position on the boundary or asking that its assigned position equals the average of its neighbors' positions. This $|V| \times |V|$ system of equations defines a *harmonic* parameterization.

The final output in Figure 4.5(b) is laid out elastically, evenly distributing vertices on the image plane. Harmonic parameterization has been extended in countless ways to enhance the quality of this result, most prominently by accounting for the lengths of the edges in E as they are realized in three-dimensional space.

4.2 SPECIAL PROPERTIES OF LINEAR SYSTEMS

The examples above provide several contexts in which linear systems of equations are used to model practical computing problems. As derived in the previous chapter, Gaussian elimination solves all of these problems in polynomial time, but it remains to be seen whether this is the fastest or most stable technique. With this question in mind, here we look more closely at the matrices from §4.1 to reveal that they have many properties in common. By deriving solution techniques specific to these classes of matrices, we will design specialized algorithms with better speed and numerical quality.

4.2.1 Positive Definite Matrices and the Cholesky Factorization

As shown in Theorem 4.1, solving the least-squares problem $A\vec{x} \approx \vec{b}$ yields a solution $\vec{x}$ satisfying the square linear system $(A^\top A)\vec{x} = A^\top \vec{b}$. Regardless of A, the matrix $A^\top A$ has a few special properties that distinguish it from arbitrary matrices.

First, $A^\top A$ is symmetric, since by the identities $(AB)^\top = B^\top A^\top$ and $(A^\top)^\top = A$,

$$(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A.$$

We can express this symmetry index-wise by writing $(A^\top A)_{ij} = (A^\top A)_{ji}$ for all indices i, j. This property implies that it is sufficient to store only the values of $A^\top A$ on or above the diagonal, since the rest of the elements can be obtained by symmetry.

Furthermore, $A^\top A$ is a *positive semidefinite* matrix, defined below:

Definition 4.1 (Positive (Semi-)Definite)**.** A matrix $B \in \mathbb{R}^{n \times n}$ is positive semidefinite if for all $\vec{x} \in \mathbb{R}^n$, $\vec{x}^\top B \vec{x} \geq 0$. B is positive definite if $\vec{x}^\top B \vec{x} > 0$ whenever $\vec{x} \neq \vec{0}$.

The following proposition relates this definition to the matrix $A^\top A$:

Proposition 4.1. For any $A \in \mathbb{R}^{m\times n}$, the matrix $A^\top A$ is positive semidefinite. Furthermore, $A^\top A$ is positive definite exactly when the columns of A are linearly independent.

Proof. We first check that $A^\top A$ is *always* positive semidefinite. Take any $\vec{x} \in \mathbb{R}^n$. Then,

$$\vec{x}^\top (A^\top A)\vec{x} = (A\vec{x})^\top (A\vec{x}) = (A\vec{x}) \cdot (A\vec{x}) = \|A\vec{x}\|_2^2 \geq 0.$$

To prove the second statement, first suppose the columns of A are linearly independent. If A were only semidefinite, then there would be an $\vec{x} \neq \vec{0}$ with $\vec{x}^\top A^\top A\vec{x} = 0$, but as shown above, this would imply $\|A\vec{x}\|_2 = 0$, or equivalently $A\vec{x} = \vec{0}$, contradicting the independence of the columns of A. Conversely, if A has linearly *dependent* columns, then there exists a $\vec{y} \neq \vec{0}$ with $A\vec{y} = \vec{0}$. In this case, $\vec{y}^\top A^\top A\vec{y} = \vec{0}^\top\vec{0} = 0$, and hence A is not positive definite. □

As a corollary, $A^\top A$ is invertible exactly when A has linearly independent columns, providing a condition to check whether a least-squares problem admits a unique solution.

Given the prevalence of the least-squares system $A^\top A\vec{x} = A^\top\vec{b}$, it is worth considering the possibility of writing faster linear solvers specially designed for this case. In particular, suppose we wish to solve a *symmetric positive definite* (SPD) system $C\vec{x} = \vec{d}$. For least-squares, we could take $C = A^\top A$ and $\vec{d} = A^\top\vec{b}$, but there also exist many systems that naturally are symmetric and positive definite without explicitly coming from a least-squares model. We could solve the system using Gaussian elimination or LU factorization, but given the additional structure on C we can do somewhat better.

Aside 4.1 (Block matrix notation)**.** Our construction in this section will rely on *block matrix* notation. This notation builds larger matrices out of smaller ones. For example, suppose $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{m\times k}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times k}$. Then, we could construct a larger matrix by writing:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \in \mathbb{R}^{(m+p)\times(n+k)}.$$

This "block matrix" is constructed by concatenation. Block matrix notation is convenient, but we must be careful to concatenate matrices with dimensions that match. The mechanisms of matrix algebra generally extend to this case, e.g.,

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

We will proceed without checking these identities explicitly, but as an exercise it is worth double-checking that they are true.

We can deconstruct the symmetric positive-definite matrix $C \in \mathbb{R}^{n\times n}$ as a block matrix:

$$C = \begin{pmatrix} c_{11} & \vec{v}^\top \\ \vec{v} & \tilde{C} \end{pmatrix}$$

where $c_{11} \in \mathbb{R}$, $\vec{v} \in \mathbb{R}^{n-1}$, and $\tilde{C} \in \mathbb{R}^{(n-1)\times(n-1)}$. The SPD structure of C provides the following observation:

$$\begin{aligned} 0 &< \vec{e}_1^\top C\vec{e}_1 \text{ since } C \text{ is positive definite and } \vec{e}_1 \neq \vec{0} \\ &= \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}\begin{pmatrix} c_{11} & \vec{v}^\top \\ \vec{v} & \tilde{C} \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \end{aligned}$$

$$= \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} c_{11} \\ \vec{v} \end{pmatrix}$$
$$= c_{11}.$$

By the strict inequality in the first line, we do not have to use pivoting to guarantee that $c_{11} \neq 0$ in the first step of Gaussian elimination.

Continuing with Gaussian elimination, we can apply a forward-substitution matrix E of the form

$$E = \begin{pmatrix} 1/\sqrt{c_{11}} & \vec{0}^\top \\ \vec{r} & I_{(n-1)\times(n-1)} \end{pmatrix}.$$

Here, the vector $\vec{r} \in \mathbb{R}^{n-1}$ contains forward-substitution scaling factors satisfying $r_{i-1}c_{11} = -c_{i1}$. Unlike our original construction of Gaussian elimination, we scale row 1 by $1/\sqrt{c_{11}}$ for reasons that will become apparent shortly.

By design, after forward-substitution, the form of the product EC is:

$$EC = \begin{pmatrix} \sqrt{c_{11}} & \vec{v}^\top/\sqrt{c_{11}} \\ \vec{0} & D \end{pmatrix},$$

for some $D \in \mathbb{R}^{(n-1)\times(n-1)}$.

Now, we diverge from the derivation of Gaussian elimination. Rather than moving on to the second row, to maintain symmetry, we post-multiply by $E^\top$ to obtain $ECE^\top$:

$$\begin{aligned} ECE^\top &= (EC)E^\top \\ &= \begin{pmatrix} \sqrt{c_{11}} & \vec{v}^\top/\sqrt{c_{11}} \\ \vec{0} & D \end{pmatrix} \begin{pmatrix} 1/\sqrt{c_{11}} & \vec{r}^\top \\ \vec{0} & I_{(n-1)\times(n-1)} \end{pmatrix} \\ &= \begin{pmatrix} 1 & \vec{0}^\top \\ \vec{0} & D \end{pmatrix}. \end{aligned}$$

The $\vec{0}^\top$ in the upper right follows from the construction of E as an elimination matrix. Alternatively, an easier if less direct argument is that $ECE^\top$ is symmetric, and the lower-left element of the block form for $ECE^\top$ is $\vec{0}$ by block matrix multiplication. Regardless, we have eliminated the first row *and* the first column of C! Furthermore, the remaining submatrix D is also symmetric and positive definite, as suggested in Exercise 4.2.

Example 4.6 (Cholesky factorization, initial step)**.** As a concrete example, consider the following symmetric, positive definite matrix

$$C = \begin{pmatrix} 4 & -2 & 4 \\ -2 & 5 & -4 \\ 4 & -4 & 14 \end{pmatrix}.$$

We can eliminate the first column of C using the elimination matrix E_1 defined as:

$$E_1 = \begin{pmatrix} 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \longrightarrow E_1C = \begin{pmatrix} 2 & -1 & 2 \\ 0 & 4 & -2 \\ 0 & -2 & 10 \end{pmatrix}.$$

We chose the upper left element of E_1 to be $1/2 = 1/\sqrt{4} = 1/\sqrt{c_{11}}$. Following the construction above, we can post-multiply by $E_1^\top$ to obtain:

$$E_1CE_1^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & -2 \\ 0 & -2 & 10 \end{pmatrix}.$$

The first row *and* column of this product equal the standard basis vector $\vec{e}_1 = (1, 0, 0)$.

We can repeat this process to eliminate all the rows and columns of C symmetrically. This method is *specific* to symmetric positive-definite matrices, since

- symmetry allowed us to apply the same E to both sides, and
- positive definiteness guaranteed that $c_{11} > 0$, thus implying that $1/\sqrt{c_{11}}$ exists.

Similar to LU factorization, we now obtain a factorization $C = LL^\top$ for a lower-triangular matrix L. This factorization is constructed by applying elimination matrices symmetrically using the process above, until we reach

$$E_k \cdots E_2 E_1 C E_1^\top E_2^\top \cdots E_k^\top = I_{n\times n}.$$

Then, like our construction in §3.5.1, we define L as a product of lower-triangular matrices:

$$L \equiv E_1^{-1} E_2^{-1} \cdots E_k^{-1}.$$

The product $C = LL^\top$ is known as the *Cholesky factorization* of C. If taking the square roots causes numerical issues, a related $LDL^\top$ factorization, where D is a diagonal matrix, avoids this issue and can be derived from the discussion above; see Exercise 4.6.

Example 4.7 (Cholesky factorization, remaining steps)**.** Continuing Example 4.6, we can eliminate the second row and column as follows:

$$E_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 1/2 & 1 \end{pmatrix} \longrightarrow E_2(E_1 C E_1^\top)E_2^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 9 \end{pmatrix}.$$

Rescaling brings the symmetric product to the identity matrix $I_{3\times3}$:

$$E_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \longrightarrow E_3(E_2 E_1 C E_1^\top E_2^\top)E_3^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Hence, we have shown $E_3 E_2 E_1 C E_1^\top E_2^\top E_3^\top = I_{3\times3}$. As above, define:

$$L = E_1^{-1} E_2^{-1} E_3^{-1} = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & -1 & 3 \end{pmatrix}.$$

This matrix L satisfies $LL^\top = C$.

The Cholesky factorization has many practical properties. It takes half the memory to store L from the Cholesky factorization rather than the LU factorization of C. Specifically, L has $n(n+1)/2$ nonzero elements, while the compressed storage of LU factorizations explained in §3.5.3 requires n^2 nonzeros. Furthermore, as with the LU decomposition, solving $C\vec{x} = \vec{d}$ can be accomplished using fast forward- and back-substitution. Finally, the product $LL^\top$ is symmetric and positive semidefinite regardless of L; if we factored $C = LU$ but made rounding and other mistakes, in degenerate cases the computed product $C' \approx LU$ may no longer satisfy these criteria exactly.

Code for Cholesky factorization can be very succinct. To derive a particularly compact form, we can work backward from the factorization $C = LL^\top$ now that we know such an object exists. Suppose we choose an arbitrary $k \in \{1, \ldots, n\}$ and write L in block form isolating the k-th row and column:

$$L = \begin{pmatrix} L_{11} & \vec{0} & 0 \\ \vec{\ell}_k^\top & \ell_{kk} & \vec{0}^\top \\ L_{31} & \vec{\ell}_k' & L_{33} \end{pmatrix}.$$

Here, since L is lower triangular, L_{11} and L_{33} are both lower-triangular square matrices. Applying block matrix algebra to the product $C = LL^\top$ shows:

$$\begin{aligned} C = LL^\top &= \begin{pmatrix} L_{11} & \vec{0} & 0 \\ \vec{\ell}_k^\top & \ell_{kk} & \vec{0}^\top \\ L_{31} & \vec{\ell}_k' & L_{33} \end{pmatrix} \begin{pmatrix} L_{11}^\top & \vec{\ell}_k & L_{31}^\top \\ \vec{0}^\top & \ell_{kk} & (\vec{\ell}_k')^\top \\ 0 & \vec{0} & L_{33}^\top \end{pmatrix} \\ &= \begin{pmatrix} \times & \times & \times \\ \vec{\ell}_k^\top L_{11}^\top & \vec{\ell}_k^\top \vec{\ell}_k + \ell_{kk}^2 & \times \\ \times & \times & \times \end{pmatrix}. \end{aligned}$$

We leave out values of the product that are not necessary for our derivation.

Since $C = LL^\top$, from the product above we now have $c_{kk} = \vec{\ell}_k^\top \vec{\ell}_k + \ell_{kk}^2$, or equivalently

$$\ell_{kk} = \sqrt{c_{kk} - \|\vec{\ell}_k\|_2^2},$$

where $\vec{\ell}_k \in \mathbb{R}^{k-1}$ contains the elements of the k-th row of L to the left of the diagonal. We can choose $\ell_{kk} \geq 0$, since scaling columns of L by -1 has no effect on $C = LL^\top$. Furthermore, applying $C = LL^\top$ to the middle left element of the product shows $L_{11}\vec{\ell}_k = \vec{c}_k$, where $\vec{c}_k$ contains the elements of C in the same position as $\vec{\ell}_k$. Since L_{11} is lower triangular, this system can be solved by forward-substitution for $\vec{\ell}_k$!

Synthesizing the formulas above reveals an algorithm for computing the Cholesky factorization by iterating $k = 1, 2, \ldots, n$. L_{11} will already be computed by the time we reach row k, so $\vec{\ell}_k$ can be found using forward-substitution. Then, ℓ_{kk} is computed directly using the square root formula. We provide pseudocode in Figure 4.6. As with LU factorization, this algorithm runs in $O(n^3)$ time; more specifically, Cholesky factorization takes approximately $\frac{1}{3}n^3$ operations, half the work needed for LU.

4.2.2 Sparsity

We set out in this section to identify properties of specific linear systems that can make them solvable using more efficient techniques than Gaussian elimination. In addition to positive definiteness, many linear systems of equations naturally enjoy *sparsity*, meaning that most of the entries of A in the system $A\vec{x} = \vec{b}$ are exactly zero. Sparsity can reflect particular structure in a given problem, including the following use cases:

- In image processing (e.g., §4.1.5), systems for photo editing express relationships between the values of pixels and those of their neighbors on the image grid. An image may be a point in $\mathbb{R}^p$ for p pixels, but when solving $A\vec{x} = \vec{b}$ for a new size-p image, $A \in \mathbb{R}^{p \times p}$ may have only $O(p)$ rather than $O(p^2)$ nonzeros since each row only involves a single pixel and its up/down/left/right neighbors.

- In computational geometry (e.g., §4.1.6), shapes are often expressed using collections of triangles linked together into a mesh. Equations for surface smoothing, parameterization, and other tasks link values associated with given vertex with only those at their neighbors in the mesh.

```
function CHOLESKY-FACTORIZATION($C$)
  ▷ Factors $C = LL^T$, assuming $C$ is symmetric and positive definite

  $L \leftarrow C$                      ▷ This algorithm destructively replaces $C$ with $L$
  for $k \leftarrow 1, 2, \ldots, n$
    ▷ Back-substitute to place $\vec{\ell}_k^\top$ at the beginning of row $k$
    for $i \leftarrow 1, \ldots, k-1$                ▷ Current element $i$ of $\vec{\ell}_k$
      $s \leftarrow 0$
      ▷ Iterate over $L_{11}$; $j < i$, so the iteration maintains $L_{kj} = (\vec{\ell}_k)_j$.
      for $j \leftarrow 1, \ldots, i-1 : s \leftarrow s + L_{ij}L_{kj}$
      $L_{ki} \leftarrow {(L_{ki}-s)}/{L_{ii}}$

    ▷ Apply the formula for $\ell_{kk}$
    $v \leftarrow 0$                                ▷ For computing $\|\vec{\ell}_k\|_2^2$
    for $j \leftarrow 1, \ldots, k-1 : v \leftarrow v + L_{kj}^2$
    $L_{kk} \leftarrow \sqrt{L_{kk} - v}$
  return $L$
```

Figure 4.6 Cholesky factorization for writing $C = LL^\top$, where the input C is symmetric and positive-definite and the output L is lower triangular.

- In machine learning, a *graphical model* uses a graph $G = (V, E)$ to express probability distributions over several variables. Each variable corresponds to a node $v \in V$, and edges $e \in E$ represent probabilistic dependences. Linear systems in this context often have one row per $v \in V$ with nonzeros in columns involving v and its neighbors.

If $A \in \mathbb{R}^{n \times n}$ is sparse to the point that it contains $O(n)$ rather than $O(n^2)$ nonzero values, there is no reason to store A with n^2 values. Instead, *sparse matrix* storage techniques only store the $O(n)$ nonzeros in a more reasonable data structure, e.g., a list of row/column/value triplets. The choice of a matrix data structure involves considering the likely operations that will occur on the matrix, possibly including multiplication, iteration over nonzeros, or iterating over individual rows or columns.

Unfortunately, the LU (and Cholesky) factorizations of a sparse matrix A may not result in sparse L and U matrices; this loss of structure severely limits the applicability of using these methods to solve $A\vec{x} = \vec{b}$ when A is large but sparse. Thankfully, there are many *direct sparse solvers* that produce an LU-like factorization without inducing much *fill*, or additional nonzeros; discussion of these techniques can be found in [32]. Alternatively, *iterative* techniques can obtain approximate solutions to linear systems using only multiplication by A and $A^\top$. We will derive some of these methods in Chapter 11.

4.2.3 Additional Special Structures

Certain matrices are not only sparse but also *structured*. For instance, a *tridiagonal* system of linear equations has the following pattern of nonzero values:

$$\begin{pmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}.$$

In Exercise 4.8, you will derive a special version of Gaussian elimination for dealing with this *banded* structure.

In other cases, matrices may not be sparse but might admit a sparse representation. For example, consider the *circulant* matrix:

$$\begin{pmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{pmatrix}.$$

This matrix can be stored using only the values a, b, c, d. Specialized techniques for solving systems involving this and other classes of matrices are well-studied and often are more efficient than generic Gaussian elimination.

Broadly speaking, once a problem has been reduced to a linear system $A\vec{x} = \vec{b}$, Gaussian elimination provides only one option for how to find $\vec{x}$. It may be possible to show that the matrix A for the given problem can be solved more easily by identifying special properties like symmetry, positive-definiteness, and sparsity. Interested readers should refer to the discussion in [50] for consideration of numerous cases like the ones above.

4.3 SENSITIVITY ANALYSIS

It is important to examine the matrix of a linear system to find out if it has special properties that can simplify the solution process. Sparsity, positive definiteness, symmetry, and so on provide clues to the proper algorithm to use for a particular problem. Even if a given solution strategy might work in theory, however, it is important to understand how well we can trust the output. For instance, due to rounding and other discrete effects, it might be the case that an implementation of Gaussian elimination for solving $A\vec{x} = \vec{b}$ yields a solution $\vec{x}_0$ such that $0 < \|A\vec{x}_0 - \vec{b}\|_2 \ll 1$; in other words, $\vec{x}_0$ only solves the system approximately.

One general way to understand the likelihood of error is through *sensitivity analysis*. To measure sensitivity, we ask what might happen to $\vec{x}$ if instead of solving $A\vec{x} = \vec{b}$, in reality we solve a *perturbed* system of equations $(A + \delta A)\vec{x} = \vec{b} + \delta\vec{b}$. There are two ways of viewing conclusions made by this type of analysis:

1. We may represent A and $\vec{b}$ inexactly thanks to rounding and other effects. This analysis then shows the best possible accuracy we can expect for $\vec{x}$ given the mistakes made representing the problem.

2. Suppose our solver generates an inexact approximation $\vec{x}_0$ to the solution $\vec{x}$ of $A\vec{x} = \vec{b}$. This vector $\vec{x}_0$ itself is the exact solution of a different system $A\vec{x}_0 = \vec{b}_0$ if we *define* $\vec{b}_0 \equiv A\vec{x}_0$ (be sure you understand why this sentence is not a tautology!). Understanding how changes in $\vec{x}_0$ affect changes in $\vec{b}_0$ show how sensitive the system is to slightly incorrect answers.

The discussion here is motivated by the definitions of forward and backward error in §2.2.1.

4.3.1 Matrix and Vector Norms

Before we can discuss the sensitivity of a linear system, we have to be somewhat careful to define what it means for a change $\delta\vec{x}$ to be "small." Generally, we wish to measure the length, or *norm*, of a vector $\vec{x}$. We have already encountered the two-norm of a vector:

$$\|\vec{x}\|_2 \equiv \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

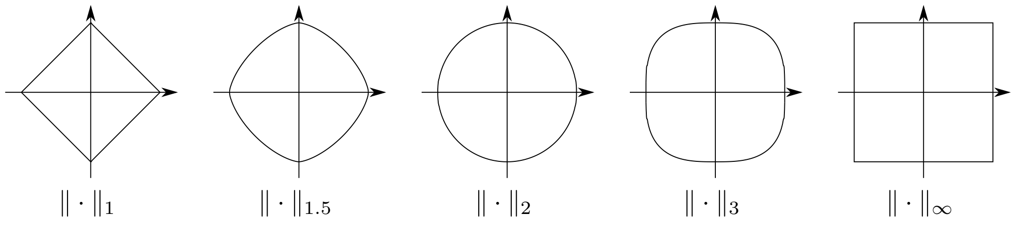


Figure 4.7 The set $\{\vec{x} \in \mathbb{R}^2 : \|\vec{x}\| = 1\}$ for different vector norms $\|\cdot\|$.

for $\vec{x} \in \mathbb{R}^n$. This norm is popular thanks to its connection to Euclidean geometry, but it is by no means the only norm on $\mathbb{R}^n$. Most generally, we define a *norm* as follows:

Definition 4.2 (Vector norm)**.** A vector norm is a function $\|\cdot\| : \mathbb{R}^n \to [0, \infty)$ satisfying the following conditions:

- $\|\vec{x}\| = 0$ if and only if $\vec{x} = \vec{0}$ ("$\|\cdot\|$ separates points").
- $\|c\vec{x}\| = |c|\|\vec{x}\|$ for all scalars $c \in \mathbb{R}$ and vectors $\vec{x} \in \mathbb{R}^n$ ("absolute scalability").
- $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$ for all $\vec{x}, \vec{y} \in \mathbb{R}^n$ ("triangle inequality").

Other than $\|\cdot\|_2$, there are many examples of norms:

- The p-norm $\|\vec{x}\|_p$, for $p \geq 1$, is given by

$$\|\vec{x}\|_p \equiv (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}.$$

  Of particular importance is the 1-norm, also known as the "Manhattan" or "taxicab" norm:

$$\|\vec{x}\|_1 \equiv \sum_{k=1}^{n} |x_k|.$$

  This norm receives its nickname because it represents the distance a taxicab drives between two points in a city where the roads only run north/south and east/west.

- The ∞-norm $\|\vec{x}\|_\infty$ is given by

$$\|\vec{x}\|_\infty \equiv \max(|x_1|, |x_2|, \cdots, |x_n|).$$

These norms are illustrated in Figure 4.7 by showing the "unit circle" $\{\vec{x} \in \mathbb{R}^2 : \|\vec{x}\| = 1\}$ for different choices of norm $\|\cdot\|$; this visualization shows that $\|\vec{v}\|_p \leq \|\vec{v}\|_q$ when $p > q$.

Despite these geometric differences, many norms on $\mathbb{R}^n$ have similar behavior. In particular, suppose we say two norms are *equivalent* when they satisfy the following property:

Definition 4.3 (Equivalent norms)**.** Two norms $\|\cdot\|$ and $\|\cdot\|'$ are *equivalent* if there exist constants c_{low} and c_{high} such that $c_{\text{low}}\|\vec{x}\| \leq \|\vec{x}\|' \leq c_{\text{high}}\|\vec{x}\|$ for all $\vec{x} \in \mathbb{R}^n$.

This condition guarantees that up to some constant factors, all norms agree on which vectors are "small" and "large." We will state without proof a famous theorem from analysis:

Theorem 4.2 (Equivalence of norms on $\mathbb{R}^n$)**.** All norms on $\mathbb{R}^n$ are equivalent.

This somewhat surprising result implies that all vector norms have the same *rough* behavior, but the choice of a norm for analyzing or stating a particular problem still can make a huge difference. For instance, on $\mathbb{R}^3$ the ∞-norm considers the vector $(1000, 1000, 1000)$ to have the same norm as $(1000, 0, 0)$, whereas the 2-norm certainly is affected by the additional nonzero values.

Since we perturb not only vectors but also matrices, we must also be able to take the norm of a matrix. The definition of a matrix norm is nothing more than Definition 4.2 with matrices in place of vectors. For this reason, we can "unroll" any matrix in $\mathbb{R}^{m\times n}$ to a vector in $\mathbb{R}^{nm}$ to adapt any vector norm to matrices. One such norm is the *Frobenius norm*

$$\|A\|_{\mathrm{Fro}} \equiv \sqrt{\sum_{i,j} a_{ij}^2}.$$

Such adaptations of vector norms, however, are not always meaningful. In particular, norms on matrices A constructed this way may not have a clear connection to the *action* of A on vectors. Since we usually use matrices to encode linear transformations, we would prefer a norm that helps us understand what happens when A is multiplied by different vectors $\vec{x}$. With this motivation, we can define the matrix norm *induced* by a vector norm as follows:

Definition 4.4 (Induced norm)**.** The matrix norm on $\mathbb{R}^{m\times n}$ *induced* by a vector norm $\|\cdot\|$ is given by

$$\|A\| \equiv \max\{\|A\vec{x}\| : \|\vec{x}\| = 1\}.$$

That is, the induced norm is the maximum length of the image of a unit vector multiplied by A.

This definition in the case $\|\cdot\| = \|\cdot\|_2$ is illustrated in Figure 4.8. Since vector norms satisfy $\|c\vec{x}\| = |c|\|\vec{x}\|$, this definition is equivalent to requiring

$$\|A\| \equiv \max_{\vec{x}\in\mathbb{R}^n\backslash\{0\}} \frac{\|A\vec{x}\|}{\|\vec{x}\|}.$$

From this standpoint, the norm of A induced by $\|\cdot\|$ is the largest achievable ratio of the norm of $A\vec{x}$ relative to that of the input $\vec{x}$.

This definition in terms of a maximization problem makes it somewhat complicated to compute the norm $\|A\|$ given a matrix A and a choice of $\|\cdot\|$. Fortunately, the matrix norms induced by many popular vector norms can be simplified. Some well-known formulae for matrix norms include the following:

- The induced one-norm of A is the maximum absolute column sum of A:

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|.$$

- The induced ∞-norm of A is the maximum absolute row sum of A:

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|.$$

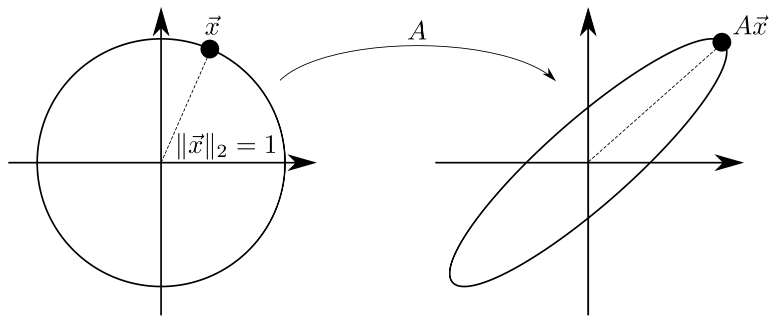


Figure 4.8 The norm $\|\cdot\|_2$ induces a matrix norm measuring the largest distortion of any point on the unit circle after applying A.

- The induced two-norm, or *spectral norm*, of $A \in \mathbb{R}^{n\times n}$ is the square root of the largest eigenvalue of $A^\top A$. That is,

$$\|A\|_2^2 = \max\{\lambda : \text{there exists } \vec{x} \in \mathbb{R}^n \text{ with } A^\top A\vec{x} = \lambda\vec{x}\}.$$

The first two norms are computable directly from the elements of A; the third will require machinery from Chapter 7.

4.3.2 Condition Numbers

Now that we have tools for measuring the action of a matrix, we can define the *condition number* of a linear system by adapting our generic definition of condition numbers from Chapter 2. In this section, we will follow the development presented in [50].

Suppose we are given a perturbation δA of a matrix A and a perturbation $\delta\vec{b}$ of the right-hand side of the linear system $A\vec{x} = \vec{b}$. For small values of ε, ignoring invertibility technicalities we can write a vector-valued function $\vec{x}(\varepsilon)$ as the solution to

$$(A + \varepsilon\cdot\delta A)\vec{x}(\varepsilon) = \vec{b} + \varepsilon\cdot\delta\vec{b}.$$

Differentiating both sides with respect to ε and applying the product rule shows:

$$\delta A\cdot\vec{x}(\varepsilon) + (A + \varepsilon\cdot\delta A)\frac{d\vec{x}(\varepsilon)}{d\varepsilon} = \delta\vec{b}.$$

In particular, when $\varepsilon = 0$ we find

$$\delta A\cdot\vec{x}(0) + A\frac{d\vec{x}}{d\varepsilon}\Big|_{\varepsilon=0} = \delta\vec{b}$$

or, equivalently,

$$\frac{d\vec{x}}{d\varepsilon}\Big|_{\varepsilon=0} = A^{-1}(\delta\vec{b} - \delta A\cdot\vec{x}(0)).$$

Using the Taylor expansion, we can write

$$\vec{x}(\varepsilon) = \vec{x}(0) + \varepsilon\vec{x}'(0) + O(\varepsilon^2),$$

where we define $\vec{x}'(0) = \frac{d\vec{x}}{d\varepsilon}\big|_{\varepsilon=0}$. Thus, we can expand the relative error made by solving the perturbed system:

$$\frac{\|\vec{x}(\varepsilon) - \vec{x}(0)\|}{\|\vec{x}(0)\|} = \frac{\|\varepsilon\vec{x}'(0) + O(\varepsilon^2)\|}{\|\vec{x}(0)\|} \text{ by the Taylor expansion above}$$

$$\begin{aligned}
&= \frac{\|\varepsilon A^{-1}(\delta\vec{b} - \delta A \cdot \vec{x}(0)) + O(\varepsilon^2)\|}{\|\vec{x}(0)\|} \text{ by the derivative we computed}\\
&\leq \frac{|\varepsilon|}{\|\vec{x}(0)\|}(\|A^{-1}\delta\vec{b}\| + \|A^{-1}\delta A \cdot \vec{x}(0))\|) + O(\varepsilon^2)\\
&\qquad \text{by the triangle inequality } \|A+B\| \leq \|A\| + \|B\|\\
&\leq |\varepsilon|\|A^{-1}\|\left(\frac{\|\delta\vec{b}\|}{\|\vec{x}(0)\|} + \|\delta A\|\right) + O(\varepsilon^2) \text{ by the identity } \|AB\| \leq \|A\|\|B\|\\
&= |\varepsilon|\|A^{-1}\|\|A\|\left(\frac{\|\delta\vec{b}\|}{\|A\|\|\vec{x}(0)\|} + \frac{\|\delta A\|}{\|A\|}\right) + O(\varepsilon^2)\\
&\leq |\varepsilon|\|A^{-1}\|\|A\|\left(\frac{\|\delta\vec{b}\|}{\|A\vec{x}(0)\|} + \frac{\|\delta A\|}{\|A\|}\right) + O(\varepsilon^2) \text{ since } \|A\vec{x}(0)\| \leq \|A\|\|\vec{x}(0)\|\\
&= |\varepsilon|\|A^{-1}\|\|A\|\left(\frac{\|\delta\vec{b}\|}{\|\vec{b}\|} + \frac{\|\delta A\|}{\|A\|}\right) + O(\varepsilon^2) \text{ since by definition } A\vec{x}(0) = \vec{b}.
\end{aligned}$$

Here we have applied some properties of induced matrix norms which follow from corresponding properties for vectors; you will check them explicitly in Exercise 4.12.

The sum $D \equiv \|\delta\vec{b}\|/\|\vec{b}\| + \|\delta A\|/\|A\|$ appearing in the last equality above encodes the magnitudes of the perturbations of δA and $\delta\vec{b}$ relative to the magnitudes of A and $\vec{b}$, respectively. From this standpoint, to first order we have bounded the relative error of perturbing the system by ε in terms of the factor $\kappa \equiv \|A\|\|A^{-1}\|$:

$$\frac{\|\vec{x}(\varepsilon) - \vec{x}(0)\|}{\|\vec{x}(0)\|} \leq |\varepsilon| \cdot D \cdot \kappa + O(\varepsilon^2)$$

Hence, the quantity κ bounds the conditioning of linear systems involving A, inspiring the following definition:

Definition 4.5 (Matrix condition number)**.** The condition number of $A \in \mathbb{R}^{n\times n}$ with respect to a given matrix norm $\|\cdot\|$ is

$$\text{cond } A \equiv \|A\|\|A^{-1}\|.$$

If A is not invertible, we take cond $A \equiv \infty$.

For nearly any matrix norm, cond $A \geq 1$ for all A. Scaling A has no effect on its condition number. Large condition numbers indicate that solutions to $A\vec{x} = \vec{b}$ are unstable under perturbations of A or $\vec{b}$.

If $\|\cdot\|$ is induced by a vector norm and A is invertible, then we have

$$\begin{aligned}
\|A^{-1}\| &= \max_{\vec{x}\neq\vec{0}} \frac{\|A^{-1}\vec{x}\|}{\|\vec{x}\|} \text{ by definition}\\
&= \max_{\vec{y}\neq\vec{0}} \frac{\|\vec{y}\|}{\|A\vec{y}\|} \text{ by substituting } \vec{y} = A^{-1}\vec{x}\\
&= \left(\min_{\vec{y}\neq\vec{0}} \frac{\|A\vec{y}\|}{\|\vec{y}\|}\right)^{-1} \text{ by taking the reciprocal.}
\end{aligned}$$

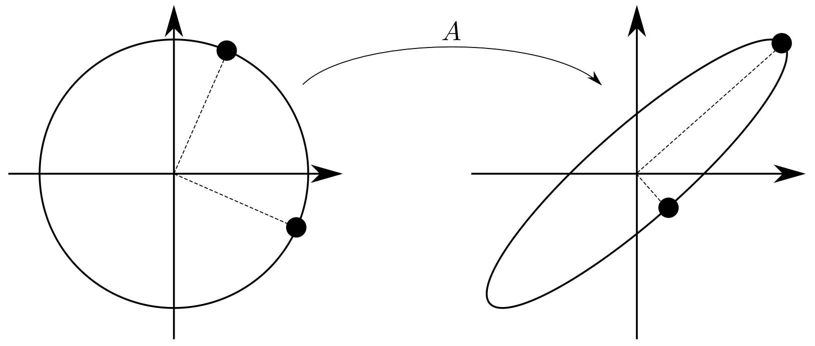


Figure 4.9 The condition number of A measures the ratio of the largest to smallest distortion of any two points on the unit circle mapped under A.

In this case, the condition number of A is given by:

$$\text{cond } A = \left(\max_{\vec{x}\neq\vec{0}} \frac{\|A\vec{x}\|}{\|\vec{x}\|}\right)\left(\min_{\vec{y}\neq\vec{0}} \frac{\|A\vec{y}\|}{\|\vec{y}\|}\right)^{-1}.$$

In other words, cond A measures the ratio of the maximum to the minimum possible stretch of a vector $\vec{x}$ under A; this interpretation is illustrated in Figure 4.9.

A desirable stability property of a system $A\vec{x} = \vec{b}$ is that if A or $\vec{b}$ is perturbed, the solution $\vec{x}$ does not change considerably. Our motivation for cond A shows that when the condition number is small, the change in $\vec{x}$ is small relative to the change in A or $\vec{b}$. Otherwise, a small change in the parameters of the linear system can cause large deviations in $\vec{x}$; this instability can cause linear solvers to make large mistakes in $\vec{x}$ due to rounding and other approximations during the solution process.

In practice, we might wish to evaluate cond A before solving $A\vec{x} = \vec{b}$ to see how successful we can expect to be in this process. Taking the norm $\|A^{-1}\|$, however, can be as difficult as computing the full inverse A^{-1}. A subtle "chicken-and-egg problem" exists here: Do we need to compute the condition number of computing matrix condition numbers? A common way out is to *bound* or *approximate* cond A using expressions that are easier to evaluate. Lower bounds on the condition number represent optimistic bounds that can be used to cull out particularly bad matrices A, while upper bounds guarantee behavior in the worst case. Condition number estimation is itself an area of active research in numerical analysis.

For example, one way to lower-bound the condition number is to apply the identity $\|A^{-1}\vec{x}\| \leq \|A^{-1}\|\|\vec{x}\|$ as in Exercise 4.12. Then, for any $\vec{x} \neq \vec{0}$ we can write $\|A^{-1}\| \geq {}^{\|A^{-1}\vec{x}\|}/_{\|\vec{x}\|}$. Thus,

$$\text{cond } A = \|A\|\|A^{-1}\| \geq \frac{\|A\|\|A^{-1}\vec{x}\|}{\|\vec{x}\|}.$$

So, we can bound the condition number by solving $A^{-1}\vec{x}$ for some vectors $\vec{x}$. The necessity of a linear solver to find $A^{-1}\vec{x}$ again creates a circular dependence on the condition number to evaluate the quality of the estimate! After considering eigenvalue problems, in future chapters we will provide more reliable estimates when $\|\cdot\|$ is induced by the two-norm.

4.4 EXERCISES

4.1 Give an example of a sparse matrix whose inverse is dense.

4.2 Show that the matrix D introduced in §4.2.1 is symmetric and positive definite.

4.3 ("Matrix calculus") The optimization problem we posed for $A \in \mathbb{R}^{2\times 2}$ in §4.1.4 is an example of a problem where the unknown is a matrix rather than a vector. These problems appear frequently in machine learning and have inspired an alternative notation for differential calculus better suited to calculations of this sort.

(a) Suppose $f : \mathbb{R}^{n\times m} \to \mathbb{R}$ is a smooth function. Justify why the gradient of f can be thought of as an $n \times m$ matrix. We will use the notation $\frac{\partial f}{\partial A}$ to notate the gradient of $f(A)$ with respect to A.

(b) Take the gradient $^{\partial}/_{\partial A}$ of the following functions, assuming $\vec{x}$ and $\vec{y}$ are constant vectors:
   (i) $\vec{x}^\top A\vec{y}$
   (ii) $\vec{x}^\top A^\top A\vec{x}$
   (iii) $(\vec{x} - A\vec{y})^\top W(\vec{x} - A\vec{y})$ for a constant, symmetric matrix W

(c) Now, suppose $X \in \mathbb{R}^{m\times n}$ is a smooth function of a scalar variable $X(t) : \mathbb{R} \to \mathbb{R}^{m\times n}$. We can notate the *differential* $\partial X \equiv X'(t)$. For matrix functions $X(t)$ and $Y(t)$, justify the following identities:
   (i) $\partial(X + Y) = \partial X + \partial Y$
   (ii) $\partial(X^\top) = (\partial X)^\top$
   (iii) $\partial(XY) = (\partial X)Y + X(\partial Y)$
   (iv) $\partial(X^{-1}) = -X^{-1}(\partial X)X^{-1}$ (see Exercise 1.13)

After establishing a dictionary of identities like the ones above, taking the derivatives of functions involving matrices becomes a far less cumbersome task. See [99] for a comprehensive reference of identities and formulas in matrix calculus.

4.4 The system of equations for A and $\vec{b}$ in §4.1.4 must be "unrolled" if we wish to use standard software for solving linear systems of equations to recover the image transformation. Define

$$A \equiv \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \text{and} \quad \vec{b} \equiv \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

We can combine all our unknowns into a vector $\vec{u}$ as follows:

$$\vec{u} \equiv \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \\ b_1 \\ b_2 \end{pmatrix}.$$

Write a matrix $M \in \mathbb{R}^{6\times 6}$ and vector $\vec{d} \in \mathbb{R}^6$ so that $\vec{u}$—and hence A and $\vec{b}$—can be recovered by solving the system $M\vec{u} = \vec{d}$ for $\vec{u}$; you can use any computable temporary variables to simplify your notation, including $\vec{x}_{\text{sum}}$, $\vec{y}_{\text{sum}}$, X, and C.

4.5 There are many ways to motivate the harmonic parameterization technique from §4.1.6. One alternative is to consider the *Dirichlet energy* of a parameterization

$$E_D[\vec{t}(\cdot)] \equiv \sum_{(v,w)\in E} \|\vec{t}(v) - \vec{t}(w)\|_2^2.$$

Then, we can write an optimization problem given boundary vertex positions $\vec{t}_0(\cdot) : B \to \mathbb{R}^2$:

$$\begin{array}{rl} \text{minimize} & E_D[\vec{t}(\cdot)] \\ \text{subject to} & \vec{t}(v) = \vec{t}_0(v)\ \forall v \in B. \end{array}$$

This optimization minimizes the Dirichlet energy $E_D[\cdot]$ over all possible parameterizations $\vec{t}(\cdot)$ with the constraint that the positions of boundary vertices $v \in B$ are fixed. Show that after minimizing this energy, interior vertices $v \in V\backslash B$ satisfy the barycenter property introduced in §4.1.6:

$$\vec{t}(v) = \frac{1}{|N(v)|}\sum_{w\in N(v)} \vec{t}(w).$$

This variational formulation connects the technique to the differential geometry of smooth maps into the plane.

4.6 A more general version of the Cholesky decomposition that does not require the computation of square roots is the LDLT decomposition.

(a) Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric and admits an LU factorization (without pivoting). Show that A can be factored $A = LDL^\top$, where D is diagonal and L is lower triangular.
Hint: Take $D \equiv UL^{-\top}$; you must show that this matrix is diagonal.

(b) Modify the construction of the Cholesky decomposition from §4.2.1 to show how a symmetric, positive-definite matrix A can be factored $A = LDL^\top$ without using any square root operations. Does your algorithm only work when A is positive definite?

4.7 Suppose $A \in \mathbb{R}^{m\times n}$ has full rank, where $m < n$. Show that taking $\vec{x} = A^\top(AA^\top)^{-1}\vec{b}$ solves the following optimization problem:

$$\begin{array}{rl} \min_{\vec{x}} & \|\vec{x}\|_2 \\ \text{subject to} & A\vec{x} = \vec{b}. \end{array}$$

Furthermore, show that taking $\alpha \to 0$ in the Tikhonov-regularized system from §4.1.3 recovers this choice of $\vec{x}$.

4.8 Suppose $A \in \mathbb{R}^{n\times n}$ is *tridiagonal*, meaning it can be written:

$$A = \begin{pmatrix} v_1 & w_1 & & & & \\ u_2 & v_2 & w_2 & & & \\ & u_3 & v_3 & w_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & u_{n-1} & v_{n-1} & w_{n-1} \\ & & & & u_n & v_n \end{pmatrix}.$$

Show that in this case the system $A\vec{x} = \vec{b}$ can be solved in $O(n)$ time. You can assume that A is *diagonally dominant*, meaning $|v_i| > |u_i| + |w_i|$ for all i.
Hint: Start from Gaussian elimination. This algorithm usually is attributed to [118].

4.9 Show how linear techniques can be used to solve the following optimization problem for $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{k\times n}, \vec{c} \in \mathbb{R}^k$:

$$\begin{aligned} \text{minimize}_{\vec{x}\in\mathbb{R}^n} & \|A\vec{x}\|_2^2 \\ \text{subject to } & B\vec{x} = \vec{c}. \end{aligned}$$

4.10 Suppose $A \in \mathbb{R}^{n\times n}$ admits a Cholesky factorization $A = LL^\top$.

(a) Show that A must be positive semidefinite.

(b) Use this observation to suggest an algorithm for checking if a matrix is positive semidefinite.

4.11 Are all matrix norms on $\mathbb{R}^{m\times n}$ equivalent? Why or why not?

4.12 For this problem, assume that the matrix norm $\|A\|$ for $A \in \mathbb{R}^{n\times n}$ is induced by a vector norm $\|\vec{v}\|$ for $\vec{v} \in \mathbb{R}^n$ (but it may be the case that $\|\cdot\| \neq \|\cdot\|_2$).

(a) For $A, B \in \mathbb{R}^{n\times n}$, show $\|A + B\| \leq \|A\| + \|B\|$.

(b) For $A, B \in \mathbb{R}^{n\times n}$ and $\vec{v} \in \mathbb{R}^n$, show $\|A\vec{v}\| \leq \|A\|\|\vec{v}\|$ and $\|AB\| \leq \|A\|\|B\|$.

(c) For $k > 0$ and $A \in \mathbb{R}^{n\times n}$, show $\|A^k\|^{1/k} \geq |\lambda|$ for any real eigenvalue λ of A.

(d) For $A \in \mathbb{R}^{n\times n}$ and $\|\vec{v}\|_1 \equiv \sum_i |v_i|$, show $\|A\|_1 = \max_j \sum_i |a_{ij}|$.

(e) Prove Gelfand's formula: $\rho(A) = \lim_{k\to\infty} \|A^k\|^{1/k}$, where $\rho(A) \equiv \max\{|\lambda_i|\}$ for eigenvalues $\lambda_1, \ldots, \lambda_m$ of A. In fact, this formula holds for any matrix norm $\|\cdot\|$.

4.13 ("Screened Poisson smoothing") Suppose we sample a function $f(x)$ at n positions $x_1, x_2, \ldots, x_n$, yielding a point $\vec{y} \equiv (f(x_1), f(x_2), \ldots, f(x_n)) \in \mathbb{R}^n$. Our measurements might be noisy, however, so a common task in graphics and statistics is to smooth these values to obtain a new vector $\vec{z} \in \mathbb{R}^n$.

(a) Provide least-squares energy terms measuring the following:

(i) The similarity of $\vec{y}$ and $\vec{z}$.

(ii) The smoothness of $\vec{z}$.
Hint: We expect $f(x_{i+1}) - f(x_i)$ to be small for smooth f.

(b) Propose an optimization problem for smoothing $\vec{y}$ using the terms above to obtain $\vec{z}$, and argue that it can be solved using linear techniques.

(c) Suppose n is very large. What properties of the matrix in 4.13b might be relevant in choosing an effective algorithm to solve the linear system?

4.14 ("Kernel trick") In this chapter, we covered techniques for linear and nonlinear *parametric* regression. Now, we will develop a least-squares technique for *nonparametic* regression that is used commonly in machine learning and vision.

(a) You can think of the least-squares problem as learning the vector $\vec{a}$ in a function $f(\vec{x}) = \vec{a} \cdot \vec{x}$ given a number of examples $\vec{x}^{(1)} \mapsto y^{(1)}, \ldots, \vec{x}^{(k)} \mapsto y^{(k)}$ and the assumption $f(\vec{x}^{(i)}) \approx y^{(i)}$. Suppose the columns of X are the vectors $\vec{x}^{(i)}$ and that $\vec{y}$ is the vector of values $y^{(i)}$. Provide the normal equations for recovering $\vec{a}$ with Tikhonov regularization.

(b) Show that $\vec{a} \in \operatorname{span}\{\vec{x}^{(1)}, \ldots, \vec{x}^{(k)}\}$ in the Tikhonov-regularized system.

(c) Thus, we can write $\vec{a} = c_1\vec{x}^{(1)} + \cdots + c_k\vec{x}^{(k)}$. Give a $k \times k$ linear system of equations satisfied by $\vec{c}$ assuming $X^\top X$ is invertible.

(d) One way to do nonlinear regression might be to write a function $\phi : \mathbb{R}^n \to \mathbb{R}^m$ and learn $f_\phi(\vec{x}) = \vec{a} \cdot \phi(\vec{x})$, where ϕ may be nonlinear. Define $K(\vec{x}, \vec{y}) = \phi(\vec{x}) \cdot \phi(\vec{y})$. Assuming we continue to use regularized least-squares as in 4.14a, give an alternative form of f_ϕ that can be computed by evaluating K rather than ϕ.
Hint: What are the elements of $X^\top X$?

(e) Consider the following formula from the Fourier transform of the Gaussian:

$$e^{-\pi(s-t)^2} = \int_{-\infty}^{\infty} e^{-\pi x^2} (\sin(2\pi sx)\sin(2\pi tx) + \cos(2\pi sx)\cos(2\pi tx))\, dx.$$

Suppose we wrote $K(x, y) = e^{-\pi(x-y)^2}$. Explain how this "looks like" $\phi(x) \cdot \phi(y)$ for some ϕ. How does this suggest that the technique from 4.14d can be generalized?

4.15 ("Discrete Fourier transform") This problem deals with complex numbers, so we will take $i \equiv \sqrt{-1}$.

(a) Suppose $\theta \in \mathbb{R}$ and $n \in \mathbb{N}$. Derive *de Moivre's formula* by induction on n:

$$(\cos\theta + i\sin\theta)^n = \cos n\theta + i\sin n\theta.$$

(b) Euler's formula uses "complex exponentials" to define $e^{i\theta} \equiv \cos\theta + i\sin\theta$. Write de Moivre's formula in this notation.

(c) Define the *primitive n-th root of unity* as $\omega_n \equiv e^{-2\pi i/n}$. The *discrete Fourier transform* matrix can be written

$$W_n \equiv \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}.$$

Show that W_n can be written in terms of a Vandermonde matrix, as defined in Example 4.3.

(d) The *complex conjugate* of $a + bi \in \mathbb{C}$, where $a, b \in \mathbb{R}$, is $\overline{a + bi} \equiv a - bi$. Show that $W_n^{-1} = W_n^*$, where $W_n^* \equiv \overline{W}_n^\top$.

(e) Suppose $n = 2^k$. In this case, show how W_n can be applied to a vector $\vec{x} \in \mathbb{C}^n$ via two applications of $W_{n/2}$ and post-processing that takes $O(n)$ time.
Note: The *fast Fourier transform* essentially uses this technique recursively to apply W_n in $O(n \log n)$ time.

(f) Suppose that A is circulant, as described in §4.2.3. Show that $W_n^* A W_n$ is diagonal.