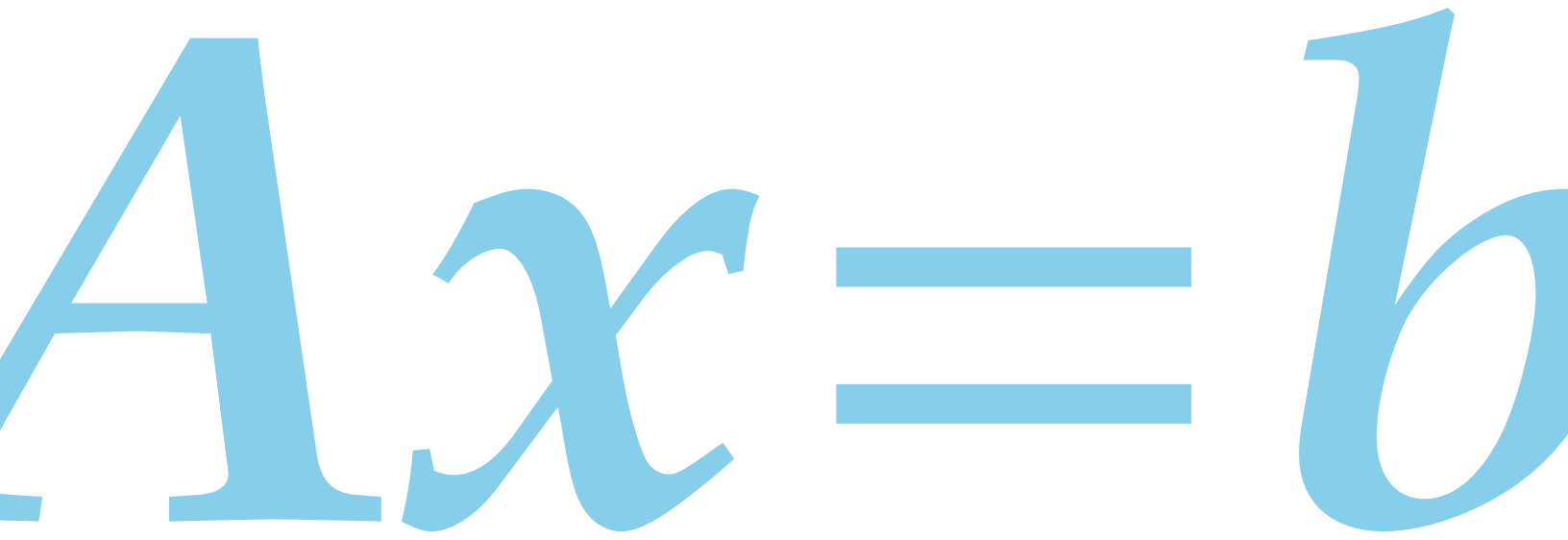
Matrix Methods *for* Computational Modeling *and* Data Analytics

Mark Embree
embree@vt.edu

Virginia Tech · 2023

version 1.3.2 [12 June 2023]



SUMMARY OF RECENT CHANGES

version 1.3.2: edits to Chapter 1.

version 1.3.1: minor typo corrections.

version 1.3: added rank definition, Rank-Nullity Theorem to Chapter 3; edited Chapter 4.

version 1.2.1: minor typo corrections and typesetting adjustments.

version 1.2: expanded subspace material in Chapter 3; moved results from Chapter 4 to Chapter 3.

version 1.1: added several marginal notes/figures in Chapters 1, 2 and 3.

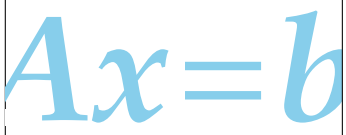
ACKNOWLEDGEMENTS

Elements of these notes (and some specific examples) were inspired by Steve Cox's *Matrix Analysis in Situ* lecture notes from CAAM 335 at Rice University. I am grateful to Steve for his inspiration and camaraderie. I also appreciate helpful suggestions and corrections from Owen Embree, Serkan Gugercin, and students who have used these notes for CMDA 3606 at Virginia Tech.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 1
Introduction

CMDA 3606 is about the linear algebra problem

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

in many wild and wonderful varieties. Most students are introduced to this equation in a sterile form, where $\mathbf{A}$ is a small, tidy matrix of obscure origin with integer entries, say

$$\mathbf{A} = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}.$$

Such examples suffice to illustrate the basics, but matrices get much more interesting when we populate $\mathbf{A}$ with meaningful entries from mathematical modeling or data science problems. In the crucible of such problems, interesting computational challenges emerge.

Linear systems of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$ often follow a simple template:

- $\mathbf{b}$ contains some quantities we can measure (i.e., a vector of *data*);
- $\mathbf{A}$ encodes some underlying mathematical/statistical model *which we also know, or can at least hypothesize, or measure*;
- $\mathbf{x}$, the *unknown*, describes the unknown parameters of the model, *which we want to discover*.

We then hope that there exists some set of parameters $\mathbf{x}$ that makes the model $\mathbf{A}\mathbf{x}$ match the data $\mathbf{b}$ (or at least approximate $\mathbf{b}$) . This context gives meaning to otherwise abstract questions like, "Does $\mathbf{A}\mathbf{x} = \mathbf{b}$ have a solution?" In data science applications, this question translates to: "Does my model exactly fit my data?"

In many realistic applications, the matrix $\mathbf{A}$ can be sufficiently large that the usual technique for solving $\mathbf{Ax} = \mathbf{b}$ problems (standard Gaussian elimination) is too slow. In this class we will briefly consider alternatives for large-scale problems.

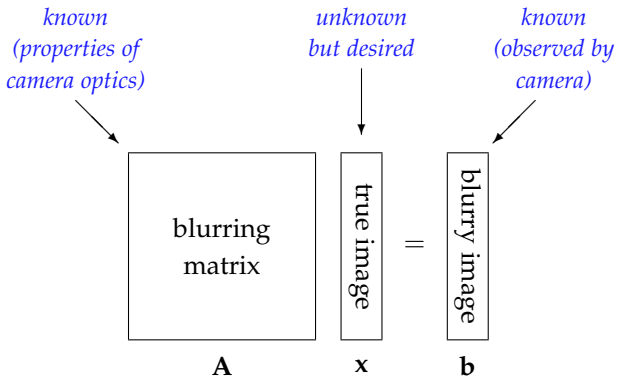
We will consider other variations of the $\mathbf{Ax} = \mathbf{b}$ problem. Sometimes we do not know the matrix $\mathbf{A}$, but we can conduct a series of physical experiments to apply known forces, $\mathbf{b}$ and *measure* the corresponding effects, $\mathbf{x}$. Can we use such experiments to *discover* $\mathbf{A}$?

This way of turning $\mathbf{Ax} = \mathbf{b}$ on its head is an example of an *inverse problem*. Often such problems are intimately connected to *least squares* and *optimization*: subjects that will occupy much of our semester.

Other situations arise where an exact solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$ exists: the theoreticians rest easy. But when you actually *compute* $\mathbf{x}$, you see that it looks like *garbage* (the polite term is *noise*): it obviously has no physical meaning!

The illustration below sketches one example of this phenomenon that arises in many problems in image processing. The vector $\mathbf{x}$ contains some image that we want to see, while the vector $\mathbf{b}$ represents the image that we can measure with our camera. The matrix $\mathbf{A}$ encodes the blurring operation that inevitably occurs when the true image $\mathbf{x}$ is mapped to the observed image $\mathbf{b}$.

More specifically, $\mathbf{b}$ is a vector of pixel values. For example, each entry in $\mathbf{b}$ might be an integer between 0 and 255, representing a shade of 8-bit grayscale (0 = black, 255 = white). To make an image into a vector, we stack each column of pixel values, one on top of the other, scanning from left to right across the image.



We can estimate $\mathbf{A}$ based on properties of the camera's optics, and calibrate it by applying the camera to a few test images for which we know the exact true image, $\mathbf{x}$.

We use the term *camera* generically; your camera might be a *telescope*: $\mathbf{A}$ can be calibrated by viewing well-studied objects, before turning your telescope to seek more interesting, unknown $\mathbf{b}$.

Here is a simple one-dimensional version of the blurring problem: scanning a UPC barcode. Consider the barcode shown below.

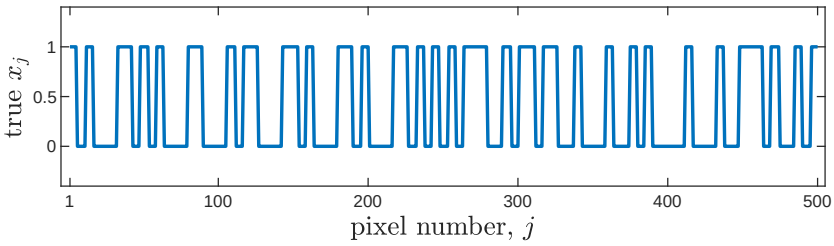
This application is mentioned by Per Christian Hansen in his book *Discrete Inverse Problems: Insight and Algorithms*, SIAM, 2010. Virginia Tech students can access the book for free: see the class website for a link.

Now take a horizontal slice of the barcode, shown in red below.

To turn this barcode into a length-n vector, we will *discretize* the red line into n pixels. Then the jth entry of $\mathbf{x}$ is determined by

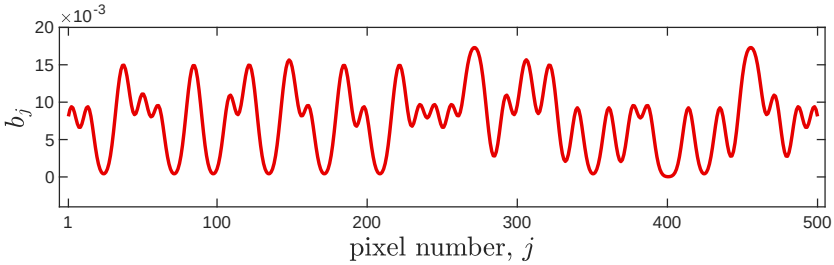
$$x_j = \begin{cases} 0, & \text{if the } j\text{th pixel is white;} \\ 1, & \text{if the } j\text{th pixel is black.} \end{cases}$$

For $n = 500$ pixels, we obtain the function shown below.



TRUE IMAGE, $\mathbf{x}$

Now we simulate the action of a supermarket barcode scanner, which detects the fuzzy version of this barcode shown in red in the plot below. This is the blurred vector, $\mathbf{b}$. From it we want to find the true barcode, $\mathbf{x}$.



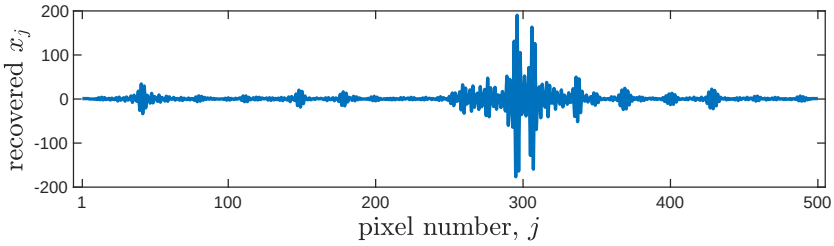
OBSERVED IMAGE, $\mathbf{b}$ (BLURRED)

Suppose we know all about the blurring operation that made this image: we know the engineering behind the optics in the scanner, and so we know the matrix $\mathbf{A}$ exactly. Indeed, this $\mathbf{A}$ is *invertible*. Knowing the scanner $\mathbf{A}$ and the scanned image $\mathbf{b}$, we should be able to compute the true, unblemished barcode:

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

In Python, `numpy.linalg.solve`.

The resulting $\mathbf{x}$ *should* (in theory) exactly match the blue barcode plot shown above. Instead, computing $\mathbf{A}^{-1}\mathbf{b}$ yields the following result: *garbage*!

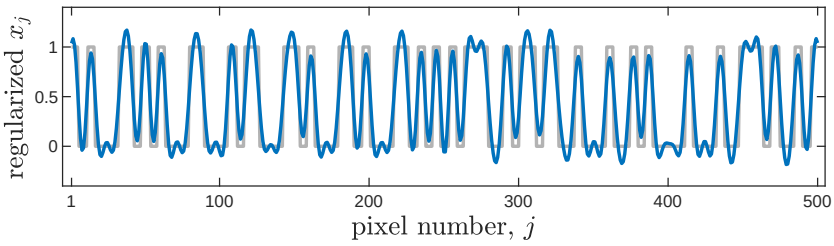


Solutions to general $\mathbf{Ax} = \mathbf{b}$ problems are not always this poor! Deblurring problems typically give $\mathbf{A}$ matrices that are especially fragile and prone to this behavior. How do you know if $\mathbf{A}$ is so vulnerable? We will find out later in the course....

RECOVERED IMAGE, $\mathbf{A}^{-1}\mathbf{b}$

What could possibly go wrong? The matrix $\mathbf{A}$ is invertible, but it is very close to a matrix that is *not invertible*, and this makes the solution of $\mathbf{Ax} = \mathbf{b}$ very sensitive to the small mistakes that occur when solving real problems in floating-point arithmetic on a computer (and to measurement noise that would normally pollute the vector $\mathbf{b}$ in applications). Unfortunately, many important, practical problems have this same structure. As a result, much research has gone into effective ways of adjusting this $\mathbf{Ax} = \mathbf{b}$ problem to discover a more robust solution. Later in the semester, we will learn about *regularization* (what statisticians call *ridge regression*). Applying this technique, we arrive at a much more satisfactory estimate of $\mathbf{x}$, shown in blue below. (The gray line underneath shows the true $\mathbf{x}$ we are trying to find.)

Modern computers use a "floating point number system" for calculations that involve real numbers. Your computer cannot represent all the (uncountably many) real numbers; instead it uses a carefully selected finite subset of the real numbers. Most of the time, this number system allows us to compute quickly and accurately (with small mistakes on the order of 10^{-16}, which we can usually ignore). However, when a problem is sensitive to small changes in the input data, you will get bad errors regardless of the cleverness of your computer's floating-point number system.



RECOVERED IMAGE (WITH REGULARIZATION)

This answer is not perfect, but to read the barcode we only need to know if our function is closer to 0 (white) or 1 (black) at a given pixel. The plot above is good enough to serve that purpose.

THIS EXAMPLE suggests that the equation $\mathbf{Ax} = \mathbf{b}$ is rather more subtle (and interesting!) than you might have thought when you encountered it in your first linear algebra class. Throughout CMDA 3606 we will try to convince you with additional examples and applications. By the semester's end, you will:

- master the singular value decomposition (SVD);
- appreciate when $\mathbf{Ax} = \mathbf{b}$ can be solved;
- apply the SVD for data analytics, including principal component analysis and recommender systems;

- understand that, when $\mathbf{A}\mathbf{x} = \mathbf{b}$ cannot be solved, we can instead find an approximation by solving

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|,$$

  which is often associated with *regression* and *inverse problems*;

Here $\|\cdot\|$ denotes a *norm*, a way of measuring the size of a vector – in this case, the mismatch between $\mathbf{A}\mathbf{x}$ and $\mathbf{b}$.

- use this approximation problem (with *regularization*) to solve inverse problems from applications, such as image deblurring;
- learn several important applications that lead to $\mathbf{A}\mathbf{x} = \mathbf{b}$ problems;
- solve optimization problems like

$$\min_{\mathbf{x}} f(\mathbf{x}),$$

  where f depends on a vector of variables, using *line search methods*, including the *stochastic gradient descent* method for large-scale problems. These technique are essential in modern deep learning applications. (Notice that $\mathbf{A}\mathbf{x} = \mathbf{b}$ is a special case, for we minimize $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$ when $\mathbf{A}\mathbf{x} = \mathbf{b}$.)

Where possible, methods will be derived rigorously, but algorithms and applications will be a constant theme. Through this course (and CMDA 3605 before it), CMDA 3606 students will acquire a significant toolkit for solving a variety of applied problems in mathematical modeling and data science. *This course teaches empowering technology.*

1.1 Prerequisites

Upon starting CMDA 3606, all students are expected to have a solid working knowledge of:

- basic matrix-vector operations;
- subspaces (especially the column space and null space of a matrix), basis, dimension, and rank;
- Gaussian elimination for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$;
- eigenvalues and eigenvectors;
- Python and NumPy.

These concepts will be reviewed just-in-time, as need for them arises throughout the semester.

Students are not expected to be expert Python programmers, but should be able to write short programs on their own. Together Python and Jupyter notebooks provide a good environment for experimenting with the concepts we will discuss throughout the semester. The instructor will provide sample codes from in-class demonstrations.

MATLAB, a commercial problem-solving environment designed around matrix computations, is a strong alternative. Julia is a modern open-source (free) environment that is gaining popularity.

1.2 Some notation and basic matrix-vector operations

We start modestly, establishing our basic conventions. Notation is the unsung hero of mathematics. The right notation clarifies, helping you see the essence of a problem. The conventions we describe have gradually evolved over linear algebra's 270 year history.

Look ahead to Figure 9.3 on page 150 to see how linear equations were expressed before matrix notation was invented.

A *vector* is a column of numbers. We write $\mathbf{v} \in \mathbb{R}^n$ to indicate that $\mathbf{v}$ is a vector of length n whose entries are *real* numbers. If the entries in $\mathbf{v}$ are *complex* numbers, we instead write $\mathbf{v} \in \mathbb{C}^n$. (In this class, most of our vectors will only contain real numbers.) In either case, we express $\mathbf{v}$ in terms of its entries:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}.$$

Vectors are typeset as bold Roman characters, while their entries (like all our scalars) are italic Roman (or Greek) letters.

Often you will need to turn a column vector into a row vector with the help of the *transpose*:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}.$$

Matrices are rectangular arrays of numbers. We write $\mathbf{A} \in \mathbb{R}^{m \times n}$ for a matrix with m rows and n columns made up of real entries; when those entries could be complex, we write $\mathbf{A} \in \mathbb{C}^{m \times n}$. In either case, we express $\mathbf{A}$ in terms of its entries:

Matrices are denoted by bold capital letters, either Roman or Greek.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}.$$

$\mathbf{A} \in \mathbb{R}^{m \times n}$ designates a matrix that has m rows and n columns. We will also refer to $\mathbf{A}$ as an *m-by-n matrix*.

We refer to $a_{j,k}$ as the "(j,k) entry of $\mathbf{A}$": that is, the entry in row j and column k.

As you become a nimble manipulator of matrices, you will find it convenient to organize these entries in different ways. For example, you might write $\mathbf{A} \in \mathbb{R}^{m \times n}$ by columns as

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix},$$

$$\mathbf{A} = \left[\begin{array}{c|c|c|c} & & & \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ & & & \end{array}\right] \in \mathbb{R}^{m \times n}$$

where each $\mathbf{a}_k$ is a vector of length m:

$$\mathbf{a}_k = \begin{bmatrix} a_{1,k} \\ a_{2,k} \\ \vdots \\ a_{m,k} \end{bmatrix}.$$

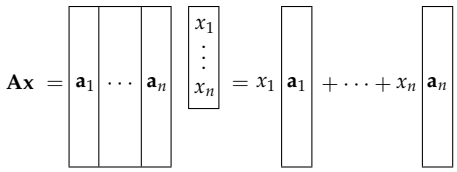
This bird's-eye view of $\mathbf{A}$ gives you a deeper appreciation for matrix-vector multiplication, for

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \sum_{k=1}^{n} x_k \mathbf{a}_k = \sum_{k=1}^{n} x_k \begin{bmatrix} a_{1,k} \\ a_{2,k} \\ \vdots \\ a_{m,k} \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix},$$

revealing $\mathbf{A}\mathbf{x}$ to be a weighted sum of the columns of $\mathbf{A}$: the entry x_k reveals how much the vector $\mathbf{a}_k$ contributes to the product $\mathbf{A}\mathbf{x}$.

Key idea: *matrix-vector multiplication is just a* linear combination *(or* weighted sum*) of the columns of* $\mathbf{A}$.



Graduating from the computational view of matrix-vector multiplication to this higher-level operational view that "$\mathbf{A}\mathbf{x}$ is a linear combination of the columns of $\mathbf{A}$" is an important step in your linear algebra maturation.

You can also build matrices from other matrices, as in

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

(assuming the dimensions match up properly). Then you can multiply against a conformally partitioned vector, simply by applying the usual rules for matrix-vector multiplication:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \\ \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \end{bmatrix}.$$

The underlying mathematical model often suggests a natural way to partition a matrix like this, where $\mathbf{x}$ and $\mathbf{y}$ represent two different kinds of variables.

Such composite matrices arise naturally in constrained multivariable optimization problems (where $\mathbf{x}$ contains the variables being optimized, and $\mathbf{y}$ contains Lagrange multipliers) and in models of incompressible fluid dynamics (where $\mathbf{x}$ contains velocities and $\mathbf{y}$ contains pressures of the fluid at points in space).

Matrix multiplication is a fundamental operation that can be interpreted in several ways, each of which is valuable in certain contexts. Consider $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$. Most mechanically, the (j,k) entry of the m-by-p matrix $\mathbf{C} := \mathbf{A}\mathbf{B}$ is given by the "row-by-column" formula:

$$c_{j,k} = \sum_{\ell=1}^{n} a_{j,\ell} b_{\ell,k}, \qquad \begin{matrix} j = 1,\ldots,m; \\ k = 1,\ldots,p. \end{matrix}$$

This formula reveals $c_{j,k}$ to be the dot product of the jth row of $\mathbf{A}$ with the kth column of $\mathbf{B}$. (We will have much more to say about dot products in the next chapter.)

Alternatively, one can view the matrix-matrix product $\mathbf{C} := \mathbf{A}\mathbf{B}$ as a series of matrix-vector products, stacked side-by-side:

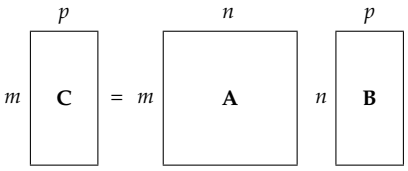
$$\begin{bmatrix} \mathbf{c}_1 & \cdots & \mathbf{c}_p \end{bmatrix} = \mathbf{A} \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_p \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{b}_1 & \cdots & \mathbf{A}\mathbf{b}_p \end{bmatrix}.$$

From this perspective, each column of $\mathbf{C}$ is the result of $\mathbf{A}$ operating on a column of $\mathbf{B}$.

Notice that the product $\mathbf{A}\mathbf{B}$ only makes sense when *the number of columns of* $\mathbf{A}$ *equals the number of rows of* $\mathbf{B}$. By taking $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$, we have implicitly insured this requirement: $\mathbf{A}$ has n columns and $\mathbf{B}$ has n rows. The product $\mathbf{C} := \mathbf{A}\mathbf{B}$ has m rows and p columns: $\mathbf{C} \in \mathbb{R}^{m\times p}$.

Remember it this way: We can multiply an m-by-n against an n-by-p because the "inner dimensions" (n) match; the "outer dimensions" (m and p) give the dimension of the product:

(m-by-n)×(n-by-p) = (m-by-p).



Several distinguished matrices merit special mention. The *zero matrix* has all entries set to zero; we write it as $\mathbf{0}$, and its dimension

will be clear from the context. The *identity matrix* is a square matrix with zeros everywhere except the main diagonal, whose entries are all ones; we denote it by $\mathbf{I}$. We shall occasionally pick out kth column of the identity, which we denote by $\mathbf{e}_k$. Thus the $n \times n$ identity is

$$\mathbf{I} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \end{bmatrix}.$$

For example, when $n = 4$,

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{1.1}$$

Note that, for all n,

$$\mathbf{Ix} = \mathbf{x}, \qquad \mathbf{IA} = \mathbf{A}, \qquad \mathbf{AI} = \mathbf{A},$$

for all vectors $\mathbf{x}$ and matrices $\mathbf{A}$. More generally, we construct *diagonal* matrices as

$$\mathbf{D} = \operatorname{diag}(d_1, \ldots, d_n) = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix} \in \mathbb{R}^{n \times n},$$

where the unspecified off-diagonal elements are zero.

A square matrix $\mathbf{B}$ for which $\mathbf{AB} = \mathbf{I}$ is called the *inverse* of $\mathbf{A}$, and is denoted by $\mathbf{A}^{-1}$. Not all matrices are invertible; for example, $\mathbf{A} = \mathbf{0}$ has no inverse. When the inverse exists, it is unique. It works on the right and the left sides of $\mathbf{A}$:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}.$$

The inverse of a product of matrices is the product of the individual inverses, *in reverse order*. For example, if both $\mathbf{A}$ and $\mathbf{B}$ are invertible, then

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}. \tag{1.2}$$

This fact is easily verified, since

$$(\mathbf{AB})(\mathbf{B}^{-1}\mathbf{A}^{-1}) = \mathbf{ABB}^{-1}\mathbf{A}^{-1} = \mathbf{AIA}^{-1} = \mathbf{AA}^{-1} = \mathbf{I}.$$

Hence $\mathbf{B}^{-1}\mathbf{A}^{-1}$ does what an inverse of $\mathbf{AB}$ is supposed to do, and since the inverse is unique, it is the only matrix that so affects $\mathbf{AB}$.

Just as we took the vector transpose, we do similarly for matrices:

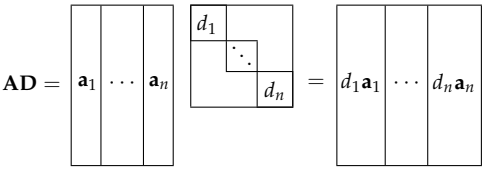
$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}, \qquad \mathbf{A}^T = \begin{bmatrix} a_{1,1} & \cdots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \cdots & a_{m,n} \end{bmatrix}.$$

$$\mathbf{0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

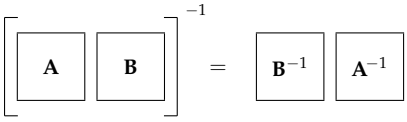
Notice this helpful trick. You can use $\mathbf{e}_k$ to extract the kth column from $\mathbf{A}$:

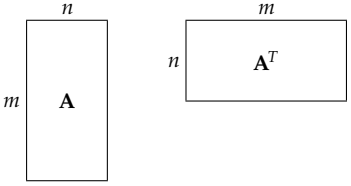
$$\mathbf{Ae}_k = \mathbf{a}_k.$$

Multiplying $\mathbf{A} \in \mathbb{R}^{m \times n}$ by a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ on the *right* side *scales the columns* of $\mathbf{A}$:



These important facts are proved in most linear algebra books/courses. We presume you are familiar with them, and only give an overview here.

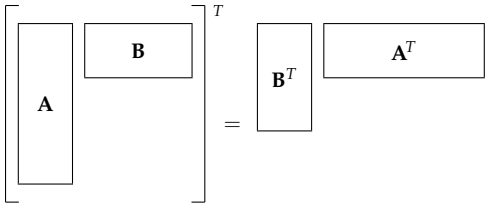




When $\mathbf{A}^T = \mathbf{A}$ we say $\mathbf{A}$ is *symmetric*. (Notice that only square matrices can be symmetric.)

The transpose distributes across addition, and distributes across a product, but reverses the order (just like inverting a product):

$$(\mathbf{A}+\mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T, \qquad (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T. \tag{1.3}$$



Perhaps it seems strange to reverse the order, but notice that it is the only sensible thing to do, from the standpoint of matrix dimensions. If $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$, then "$\mathbf{A}^T\mathbf{B}^T$" would be an n-by-m matrix times a p-by-n matrix, *which does not make sense when* $m \neq p$. In contrast, $\mathbf{B}^T\mathbf{A}^T$ is a p-by-n matrix times an n-by-m matrix, which is always defined regardless of the values of m, n, and p.

Key concept. To invert or transpose a product of matrices, *reverse the order and distributed the operation*:

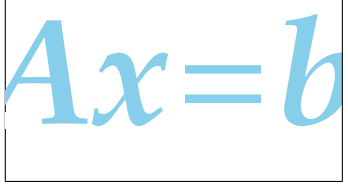
$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}, \qquad (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T.$$

You can recursively apply this idea to handle longer strings of matrices. For example, $(\mathbf{ABCD})^T = \mathbf{D}^T\mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 2
The Geometry of Vector Spaces

Before plunging into fancy matrix factorizations, we must first understand the basic geometry of the vector spaces that such matrices will operate upon.

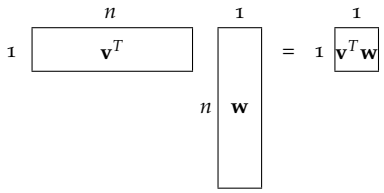
2.1 Inner and outer products, vector norms

We often multiply vectors together in two special ways.

Definition 2.1. *The* inner product *(or* dot product*) of* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *is the scalar*

$$\mathbf{v}^T\mathbf{w} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \sum_{j=1}^{n} v_j w_j \in \mathbb{R}^{1\times 1} = \mathbb{R}.$$

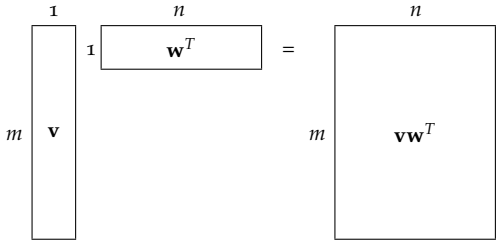
The *inner product* $\mathbf{v}^T\mathbf{w}$ can be viewed as the product of a $1 \times n$ matrix with an $n \times 1$ matrix, giving a 1×1 result.



Definition 2.2. *The* outer product *of* $\mathbf{v} \in \mathbb{R}^m$ *and* $\mathbf{w} \in \mathbb{R}^n$ *is the* $m \times n$ *matrix*

$$\mathbf{v}\mathbf{w}^T = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix} = \begin{bmatrix} v_1 w_1 & \cdots & v_1 w_n \\ \vdots & \ddots & \vdots \\ v_m w_1 & \cdots & v_m w_n \end{bmatrix} \in \mathbb{R}^{m\times n}.$$

The *outer product* $\mathbf{v}\mathbf{w}^T$ is the product of an $m \times 1$ matrix with a $1 \times n$ matrix, giving an $m \times n$ result.



Notice that, *except for special cases*, $\mathbf{v}\mathbf{w}^T \neq \mathbf{w}\mathbf{v}^T$: the order of vectors matters for an *outer* product.

Suppose $\mathbf{v} = [2, 1, 2]^T$ and $\mathbf{w} = [1, 0, -1]^T$. The *inner product* of $\mathbf{v}$ and $\mathbf{w}$ is the scalar

$$\mathbf{v}^T\mathbf{w} = \begin{bmatrix} 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = 2 \cdot 1 + 1 \cdot 0 + 2 \cdot (-1) = 0,$$

while the *outer product* is the matrix

$$\mathbf{v}\mathbf{w}^T = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & -2 \\ 1 & 0 & -1 \\ 2 & 0 & -2 \end{bmatrix}.$$

Often we take inner and outer products of a vector with itself. For example,

$$\mathbf{v}^T\mathbf{v} = \begin{bmatrix} 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} = 2 \cdot 2 + 1 \cdot 1 + 2 \cdot 2 = 9,$$

and

$$\mathbf{v}\mathbf{v}^T = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 2 & 4 \\ 2 & 1 & 2 \\ 4 & 2 & 4 \end{bmatrix}.$$

The special outer product $\mathbf{v}\mathbf{v}^T$ is always *symmetric*: using the rule (1.3),

$$(\mathbf{v}\mathbf{v}^T)^T = (\mathbf{v}^T)^T(\mathbf{v})^T = \mathbf{v}\mathbf{v}^T.$$

Sometimes we casually speak of the "length" or "size" of a vector, meaning its number of components, or, more properly, its *dimension*. However, there is another notion of the size of a vector that depends on the magnitude of the entries of the vector. This definition generalizes the absolute value of a scalar.

Some applications motivate a different notion of "length," as described by the 1-norm ("taxi cab norm")

$$\|\mathbf{v}\|_1 = |v_1| + |v_2| + \cdots + |v_n|$$

or the ∞-norm ("max norm")

$$\|\mathbf{v}\|_\infty = \max_{1 \le j \le n} |v_j|.$$

In this course, you can always assume $\|\mathbf{v}\| = \sqrt{\mathbf{v}^T\mathbf{v}}$ unless otherwise stated.

Definition 2.3. *The* norm *of a vector* $\mathbf{v} \in \mathbb{R}^n$ *is denoted*

$$\|\mathbf{v}\| = \sqrt{\sum_{j=1}^{n} |v_j|^2} = \sqrt{\mathbf{v}^T\mathbf{v}}.$$

This definition of the norm of a vector is just the usual notion of Euclidean length you are familiar with from geometry and physics. Notice that this norm obeys some simple properties:

- $\|\mathbf{v}\| \ge 0$ for all $\mathbf{v} \in \mathbb{R}^n$;
- $\|\mathbf{v}\| = 0$ if and only if $\mathbf{v} = \mathbf{0}$;
- $\|\alpha\mathbf{v}\| = |\alpha|\|\mathbf{v}\|$ for all $\alpha \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^n$;
- $\|\mathbf{v} + \mathbf{w}\| \le \|\mathbf{v}\| + \|\mathbf{w}\|$ for all $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$.

- Length cannot be negative.
- The only vector with length zero is the zero vector, $\mathbf{v} = \mathbf{0}$.
- Scaling vectors scales their length by the same amount.
- You cannot decrease length by decomposing a vector into parts.

The first three facts follow immediately from Definition 2.3. The last fact, called the *triangle inequality*, is a bit more subtle; we will address that later in the chapter.

Key concept. Throughout this course we will often use this connection between inner products and norms:

$$\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2.$$

For the previous example $\mathbf{v} = [2, 1, 2]^T$, we found $\mathbf{v}^T\mathbf{v} = 9$, and hence

$$\|\mathbf{v}\| = \sqrt{9} = 3.$$

Norms also allow us to measure the proximity of two vectors.

Definition 2.4. *The distance between the vectors* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *is defined to be the norm of their difference:*

$$\|\mathbf{w} - \mathbf{v}\|.$$

Notice that this definition obeys some natural properties we expect a "distance" to obey: you can confirm that $\|\mathbf{w} - \mathbf{v}\| \geq 0$ for any $\mathbf{v}$ and $\mathbf{w}$, and that $\|\mathbf{w} - \mathbf{v}\| = 0$ if and only if $\mathbf{w} = \mathbf{v}$.

The *triangle inequality* also holds: for any three vectors $\mathbf{u}, \mathbf{v}, \mathbf{w}$,

$$\|\mathbf{u} - \mathbf{w}\| \leq \|\mathbf{u} - \mathbf{v}\| + \|\mathbf{v} - \mathbf{w}\|.$$

This fact is a little trickier to prove; we will return to it at the end of this chapter.

We begin our construction of vector space geometry with the concept of *orthogonality*, which extends the notion of perpendicular lines.

Definition 2.5. *Two vectors* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *are* orthogonal *provided*

$$\mathbf{v}^T\mathbf{w} = 0.$$

Often the orthogonality of $\mathbf{v}$ and $\mathbf{w}$ is expressed as

$$\mathbf{v} \perp \mathbf{w}.$$

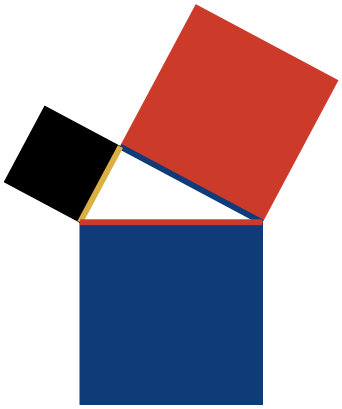


Illustration after Oliver Byrne's 1847 rendering of Euclid's *Elements*.

In this setting, the Pythagorean Theorem from classical geometry ("$A^2 + B^2 = C^2$") becomes quite simple. The proof just applies the "key concept" above.

Theorem 2.1 (Pythagorean Theorem)**.** *If* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *are orthogonal, then*

$$\|\mathbf{v} + \mathbf{w}\|^2 = \|\mathbf{v}\|^2 + \|\mathbf{w}\|^2.$$

Proof. The orthogonality of $\mathbf{v}$ and $\mathbf{w}$ implies $\mathbf{v}^T\mathbf{w} = \mathbf{w}^T\mathbf{v} = 0$, so

$$\begin{aligned}
\|\mathbf{v} + \mathbf{w}\|^2 &= (\mathbf{v} + \mathbf{w})^T(\mathbf{v} + \mathbf{w}) \\
&= (\mathbf{v}^T + \mathbf{w}^T)(\mathbf{v} + \mathbf{w}) \\
&= \mathbf{v}^T\mathbf{v} + \mathbf{v}^T\mathbf{w} + \mathbf{w}^T\mathbf{v} + \mathbf{w}^T\mathbf{w} \\
&= \mathbf{v}^T\mathbf{v} + \mathbf{w}^T\mathbf{w} \\
&= \|\mathbf{v}\|^2 + \|\mathbf{w}\|^2. \qquad \blacksquare
\end{aligned}$$

We will apply this ancient theorem at several key points this semester.

2.2 *A best approximation problem*

Many problems in this class – and in data science – involve some kind of *approximation*: finding the closest object to a given piece of data. Thus it is fitting that we should start the semester with a basic example of this kind of problem.

Suppose we have a vector $\mathbf{w} \in \mathbb{R}^n$ (the *data*) and we want to find the closest approximation to the vector in the *direction* of the vector

$\mathbf{v} \in \mathbb{R}^n$ (the *model*). (We assume that $\mathbf{v} \neq \mathbf{0}$.) The set of all vectors in the direction of $\mathbf{v}$ is given by the set

$$\text{span}\{\mathbf{v}\} = \{\alpha\mathbf{v} : \alpha \in \mathbb{R}\}.$$

This set forms a *line* in n-dimensional space. Since we can take $\alpha = 0$, this line must include the zero vector, i.e., it must go through the origin.

The notation "$\{\alpha\mathbf{v} : \alpha \in \mathbb{R}\}$" is mathematical shorthand that means "the set of all objects that have the form $\alpha\mathbf{v}$, where α is allowed to be any real number".

To solve the best approximation problem, we must clarify what we mean by "best." By that term we mean "the $\alpha\mathbf{v}$ that minimizes the distance from $\mathbf{w}$," using the notion of distance from Definition 2.4. That is, we seek that value of $\alpha \in \mathbb{R}$ that minimizes

$$\|\mathbf{w} - \alpha\mathbf{v}\|.$$

This minimizing $\alpha\mathbf{v}$ is called *the best approximation to* $\mathbf{w}$ *from* $\text{span}\{\mathbf{v}\}$. It turns out that this optimal α always has a beautiful formula.

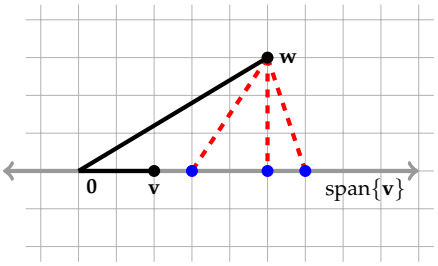


Figure 2.1: Three choices (blue dots) for approximations to $\mathbf{w}$ from $\text{span}\{\mathbf{v}\}$. The one in the middle *looks* optimal. We want to find a formula for it.

Theorem 2.2 (Best Approximation). *The best approximation to* $\mathbf{w} \in \mathbb{R}^n$ *by a vector in the direction of the nonzero* $\mathbf{v} \in \mathbb{R}^n$ *is*

$$\alpha\mathbf{v} = \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\,\mathbf{v}. \tag{2.1}$$

Where did this formula come from? We will derive it in a moment. First, make sure you understand the objects involved. Does the formula make sense? Is it always defined?

Notice that $\mathbf{v}^T\mathbf{w}$ and $\mathbf{v}^T\mathbf{v}$ are both inner products, and so $\alpha = (\mathbf{v}^T\mathbf{w})/(\mathbf{v}^T\mathbf{v})$ is just a real number. The denominator is $\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$ (see the "key concept" above), and since $\mathbf{v}$ is nonzero, $\|\mathbf{v}\| > 0$. Thus, the formula for α is well defined.

Proof. First, notice that the α that minimizes $\|\mathbf{w} - \alpha\mathbf{v}\|$ also minimizes $\|\mathbf{w} - \alpha\mathbf{v}\|^2$. We prefer to work with this latter expression, because it gets rid of the square root in the definition of the norm (Definition 2.3). Thus we have to solve

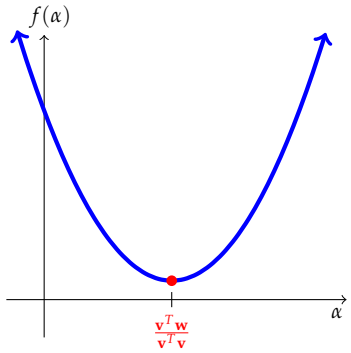
$$\min_{\alpha\in\mathbb{R}} \|\mathbf{w} - \alpha\mathbf{v}\|^2,$$

which will amount to a simple calculus problem. We proceed as in the proof of the Pythagorean Theorem, again using the "key concept" to expand

$$\begin{aligned}
\|\mathbf{w} - \alpha\mathbf{v}\|^2 &= (\mathbf{w} - \alpha\mathbf{v})^T(\mathbf{w} - \alpha\mathbf{v}) \\
&= (\mathbf{w}^T - \alpha\mathbf{v}^T)(\mathbf{w} - \alpha\mathbf{v}) \\
&= \mathbf{w}^T\mathbf{w} - \alpha\mathbf{v}^T\mathbf{w} - \alpha\mathbf{w}^T\mathbf{v} + \alpha^2\mathbf{v}^T\mathbf{v} \\
&= \mathbf{w}^T\mathbf{w} - 2\alpha\mathbf{v}^T\mathbf{w} + \alpha^2\mathbf{v}^T\mathbf{v}. \qquad (2.2)
\end{aligned}$$

The last step follows from the fact that $\mathbf{v}^T\mathbf{w} = \mathbf{w}^T\mathbf{v}$. Since $\mathbf{v}$ and $\mathbf{w}$ are given, the only variable in this last expression is α. To minimize with respect to α, define

$$f(\alpha) = \mathbf{w}^T\mathbf{w} - 2\alpha\mathbf{v}^T\mathbf{w} + \alpha^2\mathbf{v}^T\mathbf{v}$$



The function $f(\alpha)$ is the equation of a parabola in α. Since the coefficient of α^2 is $\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2 > 0$, the parabola opens up, and hence has a global minimum.

and find its minimum. To do so, compute

$$f'(\alpha) = -2\mathbf{v}^T\mathbf{w} + 2\alpha\mathbf{v}^T\mathbf{v},$$

set $f'(\alpha) = 0$, and solve for α to get

$$\alpha = \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}.$$

Indeed, this is a local minimum (since $f''(\alpha) = 2\|\mathbf{v}\|^2 > 0$), and hence the best approximation to $\mathbf{w}$ from $\text{span}\{\mathbf{v}\}$ is

$$\alpha\mathbf{v} = \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\,\mathbf{v}. \qquad \blacksquare$$

This best approximation satisfies a neat and important property. Notice that the *residual vector* (or *misfit*)

$$\mathbf{r} := \mathbf{w} - \alpha\mathbf{v} = \mathbf{w} - \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\,\mathbf{v}$$

is *orthogonal to the approximating set*. In particular,

$$\mathbf{v}^T\mathbf{r} = \mathbf{v}^T\mathbf{w} - \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\,\mathbf{v}^T\mathbf{v} = \mathbf{v}^T\mathbf{w} - \mathbf{v}^T\mathbf{w} = 0.$$

A similar calculation shows $\hat{\mathbf{v}}^T\mathbf{r} = 0$ for any $\hat{\mathbf{v}} \in \text{span}\{\mathbf{v}\}$. This fact is sufficiently important that we will document it here, so we can shine a light upon it later.

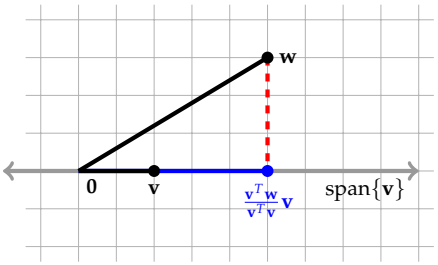


Figure 2.2: Theorem 2.2 gives a formula for the best approximation to $\mathbf{w}$ from $\text{span}\{\mathbf{v}\}$. The residual/misfit (red dashed line) is *orthogonal* to $\text{span}\{\mathbf{v}\}$.

Corollary 2.1. *The residual vector between* $\mathbf{w} \in \mathbb{R}^n$ *and its best approximation from* $\text{span}\{\mathbf{v}\}$ *is* orthogonal to the approximating set*:*

$$\mathbf{r} := \mathbf{w} - \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\,\mathbf{v} \;\perp\; \text{span}\{\mathbf{v}\}.$$

Example 2.1. Our first example comes from Figure 2.1. Let

$$\mathbf{v} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}.$$

Compute the optimal scaling factor

$$\alpha = \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}} = \frac{10}{4} = \frac{5}{2}$$

and then multiply this against $\mathbf{v}$ to get the best approximation to $\mathbf{w}$:

$$\alpha\mathbf{v} = \frac{5}{2}\begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}.$$

The residual

$$\mathbf{r} = \mathbf{w} - \alpha\mathbf{v} = \begin{bmatrix} 5 \\ 3 \end{bmatrix} - \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \end{bmatrix},$$

is clearly orthogonal to $\mathbf{v}$. $\blacksquare$

Example 2.2. Consider this rather different example. Suppose we have a series of samples, $w_1, w_2, \ldots, w_n \in \mathbb{R}$; you could imagine this being a time series, say, measuring the temperature in Blacksburg. We want to find the constant value $\alpha \in \mathbb{R}$ that best approximates all these values. We can set this up in the framework of Theorem 2.2 as follows. Put all the data in a column vector, and view $\alpha\mathbf{v}$ as a "model" approximating the data vector:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^n, \qquad \alpha\mathbf{v} = \alpha \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \alpha \\ \vdots \\ \alpha \end{bmatrix} \in \mathbb{R}^n.$$

Then we seek the value of α that minimizes $\|\mathbf{w} - \alpha\mathbf{v}\|$. Theorem 2.2 gives

$$\alpha = \frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}} = \frac{1 \cdot w_1 + 1 \cdot w_2 + \cdots + 1 \cdot w_n}{1 \cdot 1 + 1 \cdot 1 + \cdots + 1 \cdot 1} = \frac{w_1 + w_2 + w_n}{n},$$

which is just the *sample mean* (or *empirical mean*) of our data set.

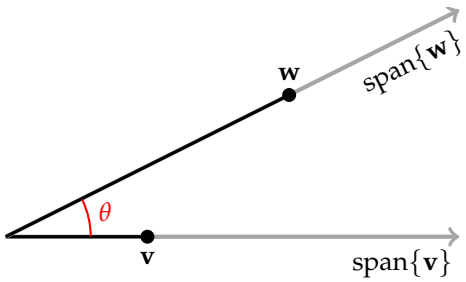
Presumably you are not surprised that the best constant approximation to $w_1, \ldots, w_n$ is the average of these values, but it is comforting to see that the theory we have developed here recovers that obvious result. ■

2.3 *Angles between vectors*

The inner product gives rise to another geometric notion, the *angle between two vectors*. In a basic linear algebra course you have likely encountered some formula relating the angle between vectors to the dot product. Where does that formula come from? *Best approximations are the key.*

Often expressed "$\mathrm{v} \cdot \mathrm{w} = \cos\theta|\mathrm{v}||\mathrm{w}|$."

We seek to measure the angle $\theta = \angle(\mathbf{v}, \mathbf{w})$ between the vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, shown in red in the picture below.

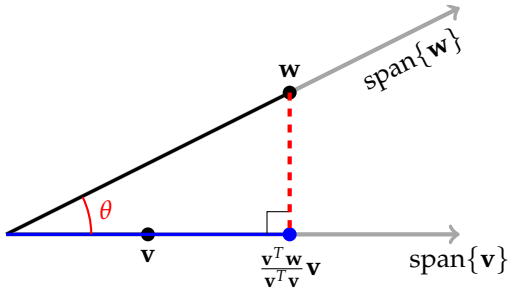


Notice that the angle is determined by the *direction* of the vector, not its length. We could imagine defining $\cos\theta$ using the familiar

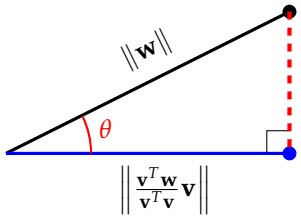
formula

$$\cos\theta = \frac{\text{LENGTH OF ADJACENT SIDE}}{\text{LENGTH OF HYPOTENUSE}}.$$

Remember from geometry that this formula holds for *right triangles*. Corollary 2.1 suggests how we can do this: if we compute the best approximation to $\mathbf{w}$ from $\text{span}\{\mathbf{v}\}$, the residual will form a right angle with the best approximation, as in the figure below.



To apply the cosine formula, we only need to compute the length of the adjacent and hypotenuse sides of the right triangle formed by this best approximation.



The definition of the cosine then gives

$$\cos\theta = \frac{\text{LENGTH OF ADJACENT SIDE}}{\text{LENGTH OF HYPOTENUSE}} = \frac{\left\|\frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\mathbf{v}\right\|}{\|\mathbf{w}\|}.$$

Since $(\mathbf{v}^T\mathbf{w})/(\mathbf{v}^T\mathbf{v})$ is a scalar, we can pull its absolute value out of the norm in the numerator, giving

$$\cos\theta = \left|\frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\right| \frac{\|\mathbf{v}\|}{\|\mathbf{w}\|} = \frac{|\mathbf{v}^T\mathbf{w}|}{\|\mathbf{v}\|^2}\frac{\|\mathbf{v}\|}{\|\mathbf{w}\|} = \frac{|\mathbf{v}^T\mathbf{w}|}{\|\mathbf{v}\|\|\mathbf{w}\|}.$$

Recall the scaling property of vector norms from page 11: for any $\alpha \in \mathbb{R}$, $\|\alpha\mathbf{v}\| = |\alpha|\|\mathbf{v}\|$.

The only difference between this formula and the one you learned in a first linear algebra class might be the absolute value bars on $|\mathbf{v}^T\mathbf{w}|$. With the absolute value bars, the angle between $\mathbf{v}$ and $\mathbf{w}$ will always be acute (i.e., $\theta \in [0, \pi/2]$), which makes particular sense if we regard θ as the (smallest) angle between the *subspaces* $\text{span}\{\mathbf{v}\}$ and $\text{span}\{\mathbf{w}\}$.

Definition 2.6. *The* angle *between the nonzero vectors* $\mathbf{v}$ *and* $\mathbf{w} \in \mathbb{R}^n$ *is*

$$\cos\angle(\mathbf{v},\mathbf{w}) = \frac{|\mathbf{v}^T\mathbf{w}|}{\|\mathbf{v}\|\|\mathbf{w}\|}. \tag{2.3}$$

If $\mathbf{v}$ and $\mathbf{w}$ are nonzero, then orthogonality is equivalent to $\cos\angle(\mathbf{v},\mathbf{w}) = 0$, i.e., $\angle(\mathbf{v},\mathbf{w}) = \pi/2$. Thus two vectors are orthogonal if they form a right angle.

Our next result ensures that

$$0 \leq \frac{|\mathbf{v}^T\mathbf{w}|}{\|\mathbf{v}\|\,\|\mathbf{w}\|} \leq 1,$$

and hence the right hand-side of (2.3) takes reasonable values for $\cos\angle(\mathbf{v},\mathbf{w})$.

Theorem 2.3 (Cauchy–Schwarz Inequality)**.** *For any* $\mathbf{v},\mathbf{w} \in \mathbb{R}^n$,

$$|\mathbf{v}^T\mathbf{w}| \leq \|\mathbf{v}\|\,\|\mathbf{w}\|.$$

Simple though it may seem, the Cauchy–Schwarz inequality is powerful and widely applicable. Its diverse proofs point to many directions in mathematics, forming the subject of an entire book:

J. Michael Steele. *The Cauchy–Schwarz Master Class*. Cambridge University Press, Cambridge, 2004

Proof. We can prove this result by adapting an argument that already appeared in the proof of the Best Approximation Theorem (Theorem 2.2). For any real number α, recall that equation (2.2) gives

$$\|\mathbf{w} - \alpha\mathbf{v}\|^2 = \|\mathbf{w}\|^2 + 2\alpha\,\mathbf{v}^T\mathbf{w} + \alpha^2\|\mathbf{v}\|^2,$$

and since the left-hand side is a norm, it cannot be negative:

$$0 \leq \|\mathbf{w}\|^2 + 2\alpha\,\mathbf{v}^T\mathbf{w} + \alpha^2\|\mathbf{v}\|^2.$$

The right-hand side is a quadratic equation in α, describing a parabola that opens up, and can never be negative. That means the quadratic cannot have a pair of distinct real roots (otherwise, the parabola would be negative in between them), and so the *discriminant* (the "$B^2 - 4AC$" part of the quadratic formula) must be non-positive. Thus

$$(2\mathbf{v}^T\mathbf{w})^2 - 4\|\mathbf{v}\|^2\|\mathbf{w}\|^2 \leq 0.$$

Recall the quadratic equation, which gives the roots of a quadratic equation $Ax^2 + Bx + C = 0$ as

$$\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.$$

In our case, $A = \|\mathbf{v}\|^2$, $B = 2\mathbf{v}^T\mathbf{w}$, and $C = \|\mathbf{w}\|^2$, and $B^2 - 4AC \leq 0$.

Rearrange this equation, divide by 4, and take square roots to get the required inequality:

$$|\mathbf{v}^T\mathbf{w}| \leq \|\mathbf{v}\|\,\|\mathbf{w}\|. \qquad \blacksquare$$

The Cauchy–Schwarz inequality enables a simple proof of the fundamental *triangle inequality*, which we encountered earlier in this chapter.

Theorem 2.4 (Triangle Inequality)**.** *For any vectors* $\mathbf{v},\mathbf{w} \in \mathbb{R}^n$,

$$\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|.$$

Proof. As in the proof of the Pythagorean Theorem, begin by expanding $\|\mathbf{v} + \mathbf{w}\|^2$ to obtain

$$\|\mathbf{v} + \mathbf{w}\|^2 = \|\mathbf{v}\|^2 + 2\mathbf{v}^T\mathbf{w} + \|\mathbf{w}\|^2.$$

Now since $\mathbf{v}^T\mathbf{w} \leq |\mathbf{v}^T\mathbf{w}|$, we have

$$\|\mathbf{v} + \mathbf{w}\|^2 \leq \|\mathbf{v}\|^2 + 2|\mathbf{v}^T\mathbf{w}| + \|\mathbf{w}\|^2.$$

Apply the Cauchy–Schwarz inequality to obtain

$$\begin{aligned}\|\mathbf{v}+\mathbf{w}\|^2 &\leq \|\mathbf{v}\|^2 + 2\|\mathbf{v}\|\,\|\mathbf{w}\| + \|\mathbf{w}\|^2 \\ &= \big(\|\mathbf{v}\| + \|\mathbf{w}\|\big)^2.\end{aligned}$$

Take the square root of both sides to obtain the result. ∎

2.4 *Projectors: surgical instruments of linear algebra*

We seek a deeper appreciation of the best approximation formula (2.1),

$$\frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}}\mathbf{v} \in \mathbb{R}^n.$$

This expression has the form "scalar ($\mathbf{v}^T\mathbf{w}/\mathbf{v}^T\mathbf{v}$) times vector ($\mathbf{v}$)", which of course gives us a vector.

Let us manipulate this expression a little bit. Move the scalar onto the right side of the vector (which we can do with scalars) to get

$$\mathbf{v}\frac{\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}} = \frac{\mathbf{v}\mathbf{v}^T\mathbf{w}}{\mathbf{v}^T\mathbf{v}} = \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\mathbf{w}.$$

As these equivalent expressions hint, we can perform the multiplications here in any order we like. For example, we can start by multiplying $\mathbf{v}\mathbf{v}^T$ – which is an *outer product*, an $n \times n$ matrix. Then we can divide the entries of that matrix by the scalar $\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$. Finally, we multiply this matrix $(\mathbf{v}\mathbf{v}^T)/(\mathbf{v}^T\mathbf{v})$ against the vector $\mathbf{w}$. This gives the equivalent formula

$$\left(\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\right)\mathbf{w}.$$

Forming that matrix $\mathbf{v}\mathbf{v}^T$ would be a very inefficient way to implement this formula in code, but it gives deeper insight into the mathematics.

Notice that matrix in parentheses does not involve the vector $\mathbf{w}$. It is a general purpose matrix that maps any vector $\mathbf{w}$ to its best approximation in $\text{span}\{\mathbf{v}\}$. Such matrices are so special that we given them a distinguished name.

Definition 2.7. *For any nonzero vector $\mathbf{v} \in \mathbb{R}^n$, the matrix*

$$\mathbf{P}_\mathbf{v} := \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}$$

is the orthogonal projector onto $\text{span}\{\mathbf{v}\}$.
The best approximation to $\mathbf{w} \in \mathbb{R}^n$ from $\text{span}\{\mathbf{v}\}$ is thus written $\mathbf{P}_\mathbf{v}\mathbf{w}$.

Some authors write "$\text{proj}_\mathbf{v}\mathbf{w}$".

If the formula for $\mathbf{P}_\mathbf{v}$ looks strange (dividing the matrix $\mathbf{v}\mathbf{v}^T$ by the scalar $\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$), you can equivalently write

$$\mathbf{P}_\mathbf{v} = \left(\frac{\mathbf{v}}{\|\mathbf{v}\|}\right)\left(\frac{\mathbf{v}}{\|\mathbf{v}\|}\right)^T.$$

$$\mathbf{P}_\mathbf{v} = \boxed{\frac{\mathbf{v}}{\|\mathbf{v}\|}}\ \boxed{\frac{\mathbf{v}^T}{\|\mathbf{v}\|}} = \boxed{\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}}$$

In dividing the vector $\mathbf{v}$ by $\|\mathbf{v}\|$, we obtain a unit vector in $\text{span}\{\mathbf{v}\}$. Thus you can understand $\mathbf{P_v}$ as the *outer product of two unit vectors*.

Notice a few neat properties of $\mathbf{P_v}$. Try to prove them for yourself.

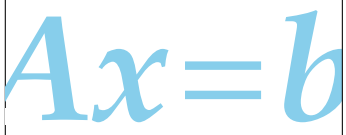
- $\mathbf{P_v}$ is *symmetric*.
- $\mathbf{P_v P_v} = \mathbf{P_v}$. We usually write $\mathbf{P_v P_v}$ as $\mathbf{P_v^2}$.
- $(\mathbf{I} - \mathbf{P_v})\mathbf{P_v} = \mathbf{0}$.
- If $\mathbf{w}$ is orthogonal to $\mathbf{v}$, then $\mathbf{P_v w} = \mathbf{0}$.
- For any $\mathbf{w} \in \mathbb{R}^n$, $(\mathbf{I} - \mathbf{P_v})\mathbf{w}$ is orthogonal to $\mathbf{v}$.

We will use projectors throughout this course. By learning how to use them nimbly, you will develop a better appreciation for key data science concepts such as regression and principal component analysis, as we shall highlight in future chapters of these notes.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 3
Orthogonalization

We need a few more tools from basic linear algebra to tackle the applications that lie ahead. This chapter addresses subspaces and bases, ultimately using projectors to orthogonalize a set of vectors. We shall see that this procedure can be viewed as a way of *factoring* a matrix into a product of two special matrices. This high-level view of the orthogonalization process is an example of the matrix-level thinking we will use to solve problems throughout this course.

3.1 Subspaces and bases

The last chapter focused on single vectors and the geometry that allowed us to compare such vectors. We also considered *one-dimensional subspaces*, such as, for nonzero $\mathbf{v} \in \mathbb{R}^m$,

$$\operatorname{span}\{\mathbf{v}\} := \{\alpha\mathbf{v} : \alpha \in \mathbb{R}\}.$$

This collection of vectors traces out a line in $\mathbb{R}^m$ that goes through the origin. To create more sophisticated models involving multiple descriptive variables, we must consider higher-dimensional subspaces of $\mathbb{R}^m$. The results in this section should be familiar from your first linear algebra course. We state them here as a reminder, and only include a few representative proofs.

Take $\alpha = 0$ to see that the origin is in $\operatorname{span}\{\mathbf{v}\}$, i.e.,

$$0\mathbf{v} = \mathbf{0} \in \operatorname{span}\{\mathbf{v}\}.$$

Definition 3.1. *The nonzero set of vectors $\mathcal{V} \subset \mathbb{R}^n$ is a* subspace *provided two properties hold:*

- *For any $\alpha \in \mathbb{R}$ and $\mathbf{v} \in \mathcal{V}$, the vector $\alpha\mathbf{v} \in \mathcal{V}$;*
- *For any $\mathbf{v}, \mathbf{w} \in \mathcal{V}$, the vector $\mathbf{v} + \mathbf{w} \in \mathcal{V}$.*

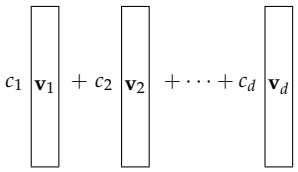
We say that "a vector space is *closed* under scalar multiplication and vector addition:" if you multiply a vector in $\mathcal{V}$ by a scalar, or add two vectors from $\mathcal{V}$ together, the result is also in $\mathcal{V}$.

We can manufacture a subspace out of any set of vectors.

Definition 3.2. *The span of the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_d \in \mathbb{R}^m$ is the set of all linear combinations (weighted sums) of these vectors:*

$$\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\} = \{c_1\mathbf{v}_1 + \cdots + c_d\mathbf{v}_d : c_1, \ldots, c_d \in \mathbb{R}\}.$$

This definition means that $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ consists of all vectors that can be written as a weighted sum of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$.



Theorem 3.1. *For any $\mathbf{v}_1, \ldots, \mathbf{v}_d \in \mathbb{R}^m$, $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ is a subspace.*

Proof. To show that $\mathcal{R}(\mathbf{A})$ is a subspace, we must verify the two properties in Definition 3.1. (i) Suppose $\mathbf{v} \in \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ and consider any $\alpha \in \mathbb{R}$. The vector $\mathbf{v}$ can be written as a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_d$:

$$\mathbf{v} = c_1\mathbf{v}_1 + \cdots + c_d\mathbf{v}_d$$

for some scalars $c_1, \ldots, c_d \in \mathbb{R}$. Then

$$\alpha\mathbf{v} = (\alpha c_1)\mathbf{v}_1 + \cdots + (\alpha c_d)\mathbf{v}_d,$$

and $\alpha\mathbf{v}$ can also be written as a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_d$. It follows that $\alpha\mathbf{v} \in \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$. We have shown that $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ is closed under scalar multiplication.

(ii) Suppose further that $\mathbf{w} \in \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$, so we can write

$$\mathbf{w} = \gamma_1\mathbf{v}_1 + \cdots + \gamma_d\mathbf{v}_d$$

for some scalars $\gamma_1, \ldots, \gamma_d \in \mathbb{R}$. Thus

$$\mathbf{v} + \mathbf{w} = (c_1 + \gamma_1)\mathbf{v}_1 + \cdots + (c_d + \gamma_d)\mathbf{v}_d,$$

and so $\mathbf{v} + \mathbf{w}$ is also a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_n$. It follows that $\mathbf{v} + \mathbf{w} \in \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$. We have thus shown that $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ is closed under vector addition.

Since $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ satisfies both requirements of Definition 3.1, we conclude that $\mathcal{R}(\mathbf{A})$ is a subspace. ∎

Definition 3.3. *Consider the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$.*

- *The column space (or range) of $\mathbf{A}$ is the set*

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m.$$

This notation means that $\mathcal{R}(\mathbf{A})$ contains all vectors of the form $\mathbf{A}\mathbf{x}$, where $\mathbf{x}$ can be any vector in $\mathbb{R}^n$. Since $\mathbf{A}\mathbf{x} \in \mathbb{R}^m$, the set $\mathcal{R}(\mathbf{A})$ contains vectors of length m, and we write $\mathcal{R}(\mathbf{A}) \subseteq \mathbb{R}^m$ to mean "$\mathcal{R}(\mathbf{A})$ is a subset of $\mathbb{R}^m$."

- *The row space of $\mathbf{A}$ is the set*

$$\mathcal{R}(\mathbf{A}^T) = \{\mathbf{A}^T\mathbf{y} : \mathbf{y} \in \mathbb{R}^m\} \subseteq \mathbb{R}^n.$$

- *The null space of $\mathbf{A}$ is the set*

$$\mathcal{N}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{0}\} \subseteq \mathbb{R}^n.$$

This notation means that $\mathcal{N}(\mathbf{A})$ contains all vectors $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{x} = \mathbf{0}$. Since such vectors $\mathbf{x}$ have length n, we write $\mathcal{N}(\mathbf{A}) \subseteq \mathbb{R}^n$ to mean "$\mathcal{N}(\mathbf{A})$ is a subset of $\mathbb{R}^n$".

Theorem 3.2. *For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the column space $\mathcal{R}(\mathbf{A})$, the row space $\mathcal{R}(\mathbf{A}^T)$, and the null space $\mathcal{N}(\mathbf{A})$ are all subspaces.*

Proof. We will only prove this theorem for $\mathcal{R}(\mathbf{A})$; see if you can adapt the proof for $\mathcal{R}(\mathbf{A}^T)$ and $\mathcal{N}(\mathbf{A})$ yourself.

To show that $\mathcal{R}(\mathbf{A})$ is a subspace, we must verify the two properties in Definition 3.1. (i) Suppose $\alpha \in \mathbb{R}$ and $\mathbf{v} \in \mathcal{R}(\mathbf{A})$. Since $\mathbf{v} \in \mathcal{R}(\mathbf{A})$, by the definition of what it means to be a vector in $\mathcal{R}(\mathbf{A})$, there must exist some vector $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{x} = \mathbf{v}$. Now consider

$$\alpha\mathbf{v} = \alpha(\mathbf{A}\mathbf{x})\mathbf{A}(\alpha\mathbf{x}).$$

Since $\alpha\mathbf{v}$ can be written as "$\mathbf{A}$ times a vector," this means that $\alpha\mathbf{v} \in \mathcal{R}(\mathbf{A})$, verifying that $\mathcal{R}(\mathbf{A})$ is closed under scalar multiplication.

(ii) Now suppose that $\mathbf{v}, \mathbf{w} \in \mathcal{R}(\mathbf{A})$. Again, by the definition of $\mathcal{R}(\mathbf{A})$, there must exist vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{x} = \mathbf{v}$ and $\mathbf{A}\mathbf{y} = \mathbf{w}$. Now consider

$$\mathbf{v} + \mathbf{w} = \mathbf{A}\mathbf{x} + \mathbf{A}\mathbf{y} = \mathbf{A}(\mathbf{x} + \mathbf{y}).$$

Since we can write $\mathbf{v} + \mathbf{w}$ as "$\mathbf{A}$ times a vector," we see $\mathbf{v} + \mathbf{w} \in \mathcal{R}(\mathbf{A})$, verifying that $\mathcal{R}(\mathbf{A})$ is closed under vector addition. Since $\mathcal{R}(\mathbf{A})$ satisfies both requirements of Definition 3.1, we conclude that $\mathcal{R}(\mathbf{A})$ is a subspace. ∎

Definition 3.4. *A set of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\} \subset \mathbb{R}^m$ is* linearly independent *provided that*

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_d\mathbf{v}_d = \mathbf{0}$$

is only possible if $c_1 = c_2 = \cdots = c_d = 0$.

Definition 3.5. *A set of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\} \subset \mathbb{R}^m$ is a* basis *for a subspace $\mathcal{V} \subseteq \mathbb{R}^m$ provided:*

- $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_d\} = \mathcal{V}$;
- *the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ is linearly independent.*

Theorem 3.3. *Every basis for the subspace $\mathcal{V}$ has the same number of vectors.*

Definition 3.6. *The number of vectors in a basis for a subspace $\mathcal{V}$ is called the* dimension *of $\mathcal{V}$. In particular, if $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ is a basis for $\mathcal{V}$, then $\dim(\mathcal{V}) = d$.*

Definition 3.7. *The* rank *of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the dimension of the column space:*

$$\text{rank}(\mathbf{A}) = \dim(\mathcal{R}(\mathbf{A})).$$

Theorem 3.4. *Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the column and row spaces have the same dimension:*

$$r := \text{rank}(\mathbf{A}) = \dim(\mathcal{R}(\mathbf{A})) = \dim(\mathcal{R}(\mathbf{A}^T)).$$

The null space $\mathcal{N}(\mathbf{A})$ has dimension $n - r$. The left null space *$\mathcal{N}(\mathbf{A}^T)$ has dimension $m - r$. Thus we can write*

$$m = \dim(\mathcal{R}(\mathbf{A})) + \dim(\mathcal{N}(\mathbf{A}^T))$$
$$n = \dim(\mathcal{R}(\mathbf{A}^T)) + \dim(\mathcal{N}(\mathbf{A})).$$

This result is often called the *Rank-Nullity Theorem*. A more complete version, known as the *Fundamental Theorem of Linear Algebra*, will be proved in Theorem 5.5.

Basis vectors provide an economical way to describe a subspace, but not all bases are created equal. Clearly the following three pairs of vectors all form bases for $\mathbb{R}^2$:

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

$$\begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}, \qquad \begin{bmatrix} \sqrt{2}/2 \\ -\sqrt{2}/2 \end{bmatrix};$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} 1, \\ .001 \end{bmatrix}.$$

For example, the vector $\mathbf{v} = [1, 1]^T$ can be written as a linear combination of all three sets of vectors. In the first two cases,

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \sqrt{2} \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix} + 0 \begin{bmatrix} \sqrt{2}/2 \\ -\sqrt{2}/2 \end{bmatrix},$$

the coefficients (in red) multiplying against the basis vectors are no bigger than $\|\mathbf{v}\| = \sqrt{2}$. In the third case,

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = -999 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 1000 \begin{bmatrix} 1 \\ .001 \end{bmatrix},$$

the coefficients -999 and 1000 are *much bigger* than $\|\mathbf{v}\|$. Small changes in $\mathbf{v}$ require significant swings in these coefficients. For example, if we change the second entry of $\mathbf{v}$ from 1 to 0.9,

$$\begin{bmatrix} 1 \\ .9 \end{bmatrix} = -899 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 900 \begin{bmatrix} 1 \\ .001 \end{bmatrix},$$

the coefficients change by 100, which is 1000 times the change in $\mathbf{v}$!

The first two bases above are special, for the norm of each vector is one, and the two basis vectors are orthogonal to each other. The vectors in the third basis form a small angle. Indeed, all these bases are linearly independent, but, we might say, some bases are more linearly independent than others.

"All animals are equal, but some animals are more equal than others."
— George Orwell, *Animal Farm* (1945)

3.2 *Orthogonalization*

This rest of this chapter focuses on orthogonalizing a set of linearly independent vectors. The main tool of this craft is the orthogonal projector onto a one dimensional subspace, as given in Definition 2.7.

Definition 3.8. *A basis* $\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ *is* orthonormal *provided*

- $\mathbf{q}_j^T \mathbf{q}_k = 0$ *for all* $j \neq k$ *(the vectors are orthogonal);*
- $\|\mathbf{q}_j\| = 1$ *for* $j = 1, \ldots, n$ *(the vectors are normalized).*

We can summarize orthonormality quite neatly if we arrange the basis vectors in the matrix

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n \end{bmatrix}.$$

Then $\mathbf{Q}^T\mathbf{Q}$ contains all possible inner products between $\mathbf{q}_j$ and $\mathbf{q}_k$:

$$\mathbf{Q}^T\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^T\mathbf{q}_1 & \mathbf{q}_1^T\mathbf{q}_2 & \cdots & \mathbf{q}_1^T\mathbf{q}_n \\ \mathbf{q}_2^T\mathbf{q}_1 & \mathbf{q}_2^T\mathbf{q}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{q}_{n-1}^T\mathbf{q}_n \\ \mathbf{q}_n^T\mathbf{q}_1 & \cdots & \mathbf{q}_n^T\mathbf{q}_{n-1} & \mathbf{q}_n^T\mathbf{q}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} = \mathbf{I},$$

since $\mathbf{q}_j^T\mathbf{q}_j = \|\mathbf{q}_j\|^2 = 1$.

Definition 3.9. *A matrix* $\mathbf{Q} \in \mathbb{R}^{n\times n}$ *is* unitary *if* $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.
If $\mathbf{Q} \in \mathbb{R}^{m\times n}$ *with* $m > n$ *and* $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, *then* $\mathbf{Q}$ *is* subunitary.

An n-by-n unitary matrix with real entries is often called an *orthogonal* matrix. The term "subunitary" (or "suborthogonal"), which follows a suggestion of Gilbert Strang, is not yet standard – but sounds tidier than "rectangular matrix with orthonormal columns".

EXERCISES

3.1. Suppose $\mathbf{Q} \in \mathbb{R}^{m\times n}$ with $m < n$. Is it possible that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$? Explain.

3.2. Suppose that $\mathbf{Q} \in \mathbb{R}^{m\times n}$ is subunitary. Show that $\mathbf{\Pi} := \mathbf{Q}\mathbf{Q}^T \in \mathbb{R}^{m\times m}$ is the sum of projectors onto the spans of the orthonormal basis vectors:

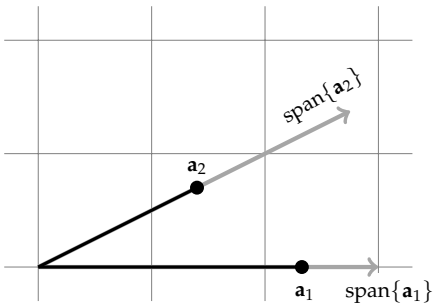
$$\mathbf{\Pi} := \mathbf{Q}\mathbf{Q}^T = \sum_{j=1}^{n} \mathbf{q}_j\mathbf{q}_j^T.$$

Now prove that $\mathbf{\Pi}$ is itself a projector, meaning that $\mathbf{\Pi}^2 = \mathbf{\Pi}$. What is the column space of $\mathbf{\Pi}$?

Recall that the *column space*) of $\mathbf{\Pi}$ is the set of all vectors of the form $\mathbf{\Pi x}$, i.e.,

$$\mathcal{R}(\mathbf{\Pi}) = \{\mathbf{\Pi x} : \mathbf{x} \in \mathbb{R}^m\}.$$

We are now prepared to orthogonalize a set of vectors. Suppose we have a basis $\mathbf{a}_1, \ldots, \mathbf{a}_n$ for a subspace $\mathcal{V} \subset \mathbb{R}^m$. We seek an orthonormal basis $\mathbf{q}_1, \ldots, \mathbf{q}_n$ for the same subspace, which we shall build one vector at a time. The goal is to first construct a unit vector $\mathbf{q}_1$ so that



The setting: basis vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ that are neither unit length, nor orthogonal.

$$\text{span}\{\mathbf{q}_1\} = \text{span}\{\mathbf{a}_1\}$$

and then to build a unit vector $\mathbf{q}_2$ orthogonal to $\mathbf{q}_1$ so that

$$\text{span}\{\mathbf{q}_1, \mathbf{q}_2\} = \text{span}\{\mathbf{a}_1, \mathbf{a}_2\}$$

and then a unit vector $\mathbf{q}_3$ orthogonal to $\mathbf{q}_1$ and $\mathbf{q}_2$ so that

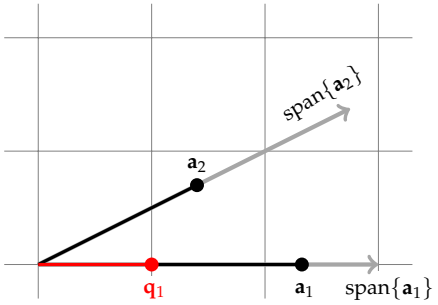
$$\text{span}\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\} = \text{span}\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$$

and so on, one $\mathbf{q}_j$ vector at a time, until we have the whole new basis:

$$\text{span}\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\} = \text{span}\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n\} = \mathcal{V}.$$

The first of these steps is easy, for we get $\mathbf{q}_1$ by normalizing $\mathbf{a}_1$:

$$\mathbf{q}_1 = \frac{1}{\|\mathbf{a}_1\|}\mathbf{a}_1.$$



First normalize $\mathbf{a}_1$ to obtain $\mathbf{q}_1$ with $\text{span}\{\mathbf{q}_1\} = \text{span}\{\mathbf{a}_1\}$.

The next step is decisive, and here we use insight gained from our study of orthogonal projectors in Chapter 2. Figuratively speaking, we must remove the part of $\mathbf{a}_2$ in the direction $\mathbf{q}_1$, leaving the portion of $\mathbf{a}_2$ orthogonal to $\mathbf{q}_1$. Since $\mathbf{q}_1$ is a unit vector,

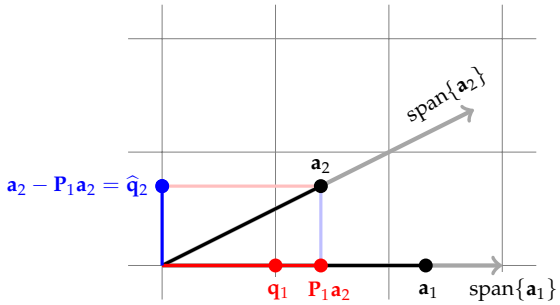
$$\mathbf{P}_1 := \mathbf{q}_1\mathbf{q}_1^T$$

is an orthogonal projector onto $\text{span}\{\mathbf{q}_1\} = \text{span}\{\mathbf{a}_1\}$, so

$$\mathbf{P}_1\mathbf{a}_2 = \mathbf{q}_1\mathbf{q}_1^T\mathbf{a}_2$$

is the part of $\mathbf{a}_2$ in the direction $\mathbf{q}_1$. Remove that from $\mathbf{a}_2$ to get

$$\widehat{\mathbf{q}}_2 := \mathbf{a}_2 - \mathbf{P}_1\mathbf{a}_2.$$



Compute $\widehat{\mathbf{q}}_2 := \mathbf{a}_2 - \mathbf{P}_1\mathbf{a}_2$, the portion of $\mathbf{a}_2$ that is orthogonal to $\mathbf{q}_1$.

Since

$$\mathbf{P}_1\widehat{\mathbf{q}}_2 = \mathbf{P}_1(\mathbf{a}_2 - \mathbf{P}_1\mathbf{a}_2) = \mathbf{0},$$

spot that $\widehat{\mathbf{q}}_2 \in \mathcal{N}(\mathbf{P}_1)$, which is orthogonal to $\mathcal{R}(\mathbf{P}_1) = \text{span}\{\mathbf{q}_1\}$, so $\widehat{\mathbf{q}}_2 \perp \mathbf{q}_1$. If you prefer a less high-falutin' explanation, just compute

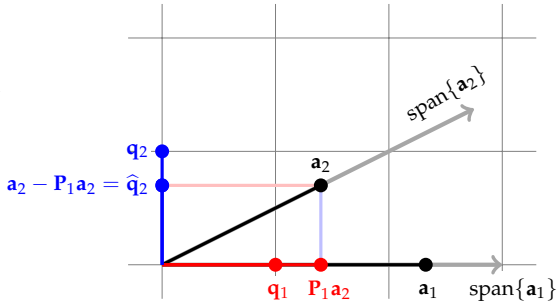
$$\mathbf{q}_1^T\widehat{\mathbf{q}}_2 = \mathbf{q}_1^T\mathbf{a}_2 - \mathbf{q}_1^T\mathbf{q}_1\mathbf{q}_1^T\mathbf{a}_2 = \mathbf{q}_1^T\mathbf{a}_2 - \mathbf{q}_1^T\mathbf{a}_2 = 0.$$

So we have constructed a vector $\widehat{\mathbf{q}}_2$ orthogonal to $\mathbf{q}_1$ such that

$$\text{span}\{\mathbf{q}_1, \widehat{\mathbf{q}}_2\} = \text{span}\{\mathbf{a}_1, \mathbf{a}_2\}.$$

Since we want not just an orthogonal basis, but an *orthonormal* basis, we adjust $\widehat{\mathbf{q}}_2$ by scaling it to become the unit vector

$$\mathbf{q}_2 := \frac{1}{\|\widehat{\mathbf{q}}_2\|}\widehat{\mathbf{q}}_2.$$



Normalize $\mathbf{q}_2 = \widehat{\mathbf{q}}_2/\|\widehat{\mathbf{q}}_2\|$ to get a unit vector in the $\widehat{\mathbf{q}}_2$ direction.

The orthogonalization process for subsequent vectors follows the same template. To construct $\mathbf{q}_3$, we first remove from $\mathbf{a}_3$ its components in the $\mathbf{q}_1$ and $\mathbf{q}_2$ directions:

$$\widehat{\mathbf{q}}_3 := \mathbf{a}_3 - \mathbf{P}_1\mathbf{a}_3 - \mathbf{P}_2\mathbf{a}_3,$$

where $\mathbf{P}_2 = \mathbf{q}_2\mathbf{q}_2^T$, where for $j = 1, 2$,

$$\begin{aligned}\mathbf{q}_j^T\widehat{\mathbf{q}}_3 &= \mathbf{q}_j^T\mathbf{a}_3 - \mathbf{q}_j^T\mathbf{q}_1\mathbf{q}_1^T\mathbf{a}_3 - \mathbf{q}_j^T\mathbf{q}_2\mathbf{q}_2^T\mathbf{a}_3 \\ &= \mathbf{q}_j^T\mathbf{a}_3 - \mathbf{q}_j^T\mathbf{a}_3 = 0,\end{aligned}$$

using the orthonormality of $\mathbf{q}_1$ and $\mathbf{q}_2$. Normalize $\widehat{\mathbf{q}}_3$ to get

$$\mathbf{q}_3 := \frac{1}{\|\widehat{\mathbf{q}}_3\|}\widehat{\mathbf{q}}_3.$$

Now the general pattern should be evident. For future vectors, construct

$$\widehat{\mathbf{q}}_{k+1} = \mathbf{a}_{k+1} - \sum_{j=1}^{k}\mathbf{P}_j\mathbf{a}_{k+1}, \quad \text{with } \mathbf{P}_j := \mathbf{q}_j\mathbf{q}_j^T,$$

then normalize

$$\mathbf{q}_{k+1} = \frac{1}{\|\widehat{\mathbf{q}}_{k+1}\|}\widehat{\mathbf{q}}_{k+1},$$

giving

$$\mathbf{q}_{k+1} \perp \operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_k\},$$

which extends the orthonormal basis in one more direction:

$$\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_k, \mathbf{q}_{k+1}\} = \operatorname{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_k, \mathbf{a}_{k+1}\}.$$

This algorithm is known as the Gram–Schmidt *process*.

EXERCISES

3.3. Under what circumstances will this procedure *break down*? That is, will you ever divide by zero when trying to construct $\mathbf{q}_{k+1} = \widehat{\mathbf{q}}_{k+1}/\|\widehat{\mathbf{q}}_{k+1}\|$?

3.4. Suppose the Gram–Schmidt process does not break down, but that $\|\widehat{\mathbf{q}}_k\| \ll \|\mathbf{a}_k\|$ for some k. What does this situation imply about how $\mathbf{a}_k$ relates to $\mathbf{a}_1, \ldots, \mathbf{a}_{k-1}$?

The notation "$\ll$" means "much less than."

3.5. Show that $\mathbf{\Pi}_k := \mathbf{P}_1 + \cdots + \mathbf{P}_k$ is an orthogonal projector, i.e., show $\mathbf{\Pi}_k^2 = \mathbf{\Pi}_k$. What space does $\mathbf{\Pi}_k$ project onto? (That is, what is the column space of $\mathbf{\Pi}_k$?) With this notation, explain how each Gram–Schmidt step can be expressed compactly as

$$\mathbf{q}_{k+1} = \frac{(\mathbf{I} - \mathbf{\Pi}_k)\mathbf{a}_{k+1}}{\|(\mathbf{I} - \mathbf{\Pi}_k)\mathbf{a}_{k+1}\|}.$$

3.3 Gram–Schmidt *is QR factorization*

The Gram–Schmidt process is a classical way to orthogonalize a basis, and you can execute the process by hand when the vectors are short (m is small) and there are few of them (n is small). However, its real power comes when we apply the technique to large collections of vectors. To do so, we need to organize the steps more systematically. Ultimately, if we arrange our original vectors as the columns of the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, then the Gram–Schmidt process can be viewed as a way of *decomposing* or *factoring* the matrix $\mathbf{A}$ into the special form $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{m\times n}$ has orthonormal columns (and hence is *subunitary*) and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is *upper triangular*, i.e., all entries below its main diagonal are zero.

Let us work through the arithmetic behind the orthogonalization of three linearly independent vectors $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$, and along the way define some quantities $r_{j,k}$ that arise in the process:

$$\begin{aligned}
\widehat{\mathbf{q}}_1 &:= \mathbf{a}_1 \\
\mathbf{q}_1 &:= \frac{1}{\|\widehat{\mathbf{q}}_1\|}\widehat{\mathbf{q}}_1 = \frac{1}{r_{1,1}}\widehat{\mathbf{q}}_1
\end{aligned} \qquad r_{1,1} = \|\widehat{\mathbf{q}}_1\|$$

$$\begin{aligned}
\widehat{\mathbf{q}}_2 &:= \mathbf{a}_2 - \mathbf{q}_1\mathbf{q}_1^T\mathbf{a}_2 \\
&= \mathbf{a}_2 - r_{1,2}\mathbf{q}_1 \\
\mathbf{q}_2 &:= \frac{1}{\|\widehat{\mathbf{q}}_2\|}\widehat{\mathbf{q}}_2 = \frac{1}{r_{2,2}}\widehat{\mathbf{q}}_2
\end{aligned} \qquad \begin{aligned} r_{1,2} &= \mathbf{q}_1^T\mathbf{a}_2 \\ r_{2,2} &= \|\widehat{\mathbf{q}}_2\| \end{aligned}$$

$$\begin{aligned}
\widehat{\mathbf{q}}_3 &:= \mathbf{a}_3 - \mathbf{q}_1\mathbf{q}_1^T\mathbf{a}_3 - \mathbf{q}_2\mathbf{q}_2^T\mathbf{a}_3 \\
&= \mathbf{a}_3 - r_{1,3}\mathbf{q}_1 - r_{2,3}\mathbf{q}_2 \\
\mathbf{q}_3 &:= \frac{1}{\|\widehat{\mathbf{q}}_3\|}\widehat{\mathbf{q}}_3 = \frac{1}{r_{3,3}}\widehat{\mathbf{q}}_3.
\end{aligned} \qquad \begin{aligned} r_{1,3} &= \mathbf{q}_1^T\mathbf{a}_3 \\ r_{2,3} &= \mathbf{q}_2^T\mathbf{a}_3 \\ r_{3,3} &= \|\widehat{\mathbf{q}}_3\| \end{aligned}$$

To summarize, we have defined

$$r_{j,k} = \begin{cases} \mathbf{q}_j^T\mathbf{a}_k, & j < k; \\ \|\widehat{\mathbf{q}}_j\|, & j = k; \\ 0, & j > k. \end{cases}$$

With this notation, three steps of the Gram–Schmidt process become:

$$\begin{aligned}
r_{1,1}\mathbf{q}_1 &= \mathbf{a}_1 \\
r_{2,2}\mathbf{q}_2 &= \mathbf{a}_2 - r_{1,2}\mathbf{q}_1 \\
r_{3,3}\mathbf{q}_3 &= \mathbf{a}_3 - r_{1,3}\mathbf{q}_1 - r_{2,3}\mathbf{q}_2,
\end{aligned}$$

The diagonal entry $r_{j,j}$ contains some very interesting information. Since

$$r_{j,j} = \|\widehat{\mathbf{q}}_j\| = \|\mathbf{a}_j - (\mathbf{P}_1 + \cdots + \mathbf{P}_{j-1})\mathbf{a}_j\|,$$

$r_{j,j}$ measures the portion of $\mathbf{a}_j$ that is *not already captured* by the directions $\mathbf{q}_1, \ldots, \mathbf{q}_{j-1}$. Consider the extreme cases.

- What does it mean when $r_{j,j} = \|\mathbf{a}_j\|$?
- What does it mean when $r_{j,j} = 0$?

or, collecting the $\mathbf{a}_j$ vectors on the left hand side:

$$\begin{aligned}\mathbf{a}_1 &= r_{1,1}\mathbf{q}_1 + 0\mathbf{q}_2 + 0\mathbf{q}_3\\ \mathbf{a}_2 &= r_{1,2}\mathbf{q}_1 + r_{2,2}\mathbf{q}_2 + 0\mathbf{q}_3\\ \mathbf{a}_3 &= r_{1,3}\mathbf{q}_1 + r_{2,3}\mathbf{q}_2 + r_{3,3}\mathbf{q}_3.\end{aligned}$$

Stack the orthonormal basis vectors as the columns of the *subunitary* matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$

$$\mathbf{Q} = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3\end{bmatrix},$$

and note that

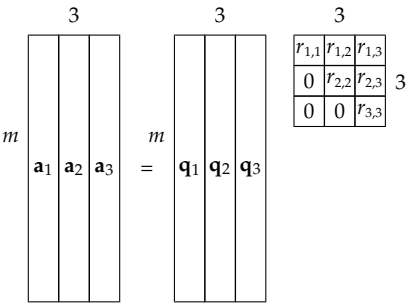
$$\mathbf{a}_1 = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3\end{bmatrix}\begin{bmatrix}r_{1,1}\\0\\0\end{bmatrix}$$

$$\mathbf{a}_2 = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3\end{bmatrix}\begin{bmatrix}r_{1,2}\\r_{2,2}\\0\end{bmatrix}$$

$$\mathbf{a}_3 = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3\end{bmatrix}\begin{bmatrix}r_{1,3}\\r_{2,3}\\r_{3,3}\end{bmatrix},$$

which we organize in matrix form as

$$\begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3\end{bmatrix} = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3\end{bmatrix}\begin{bmatrix}r_{1,1} & r_{1,2} & r_{1,3}\\0 & r_{2,2} & r_{2,3}\\0 & 0 & r_{3,3}\end{bmatrix}.$$

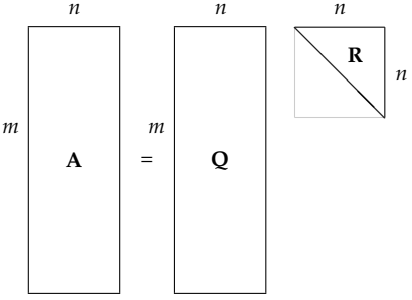


We summarize the entire process as:

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q} \in \mathbb{R}^{m\times n}$ is subunitary and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is upper triangular.

Now change your perspective: let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be any matrix with linearly independent columns, $m \geq n$. Performing the Gram–Schmidt process on those columns yields the factorization $\mathbf{A} = \mathbf{QR}$, which is known as the *QR factorization*. When $\mathbf{A}$ is a data matrix, the QR factorization gives considerable insight about that data. As we shall see, it can be used to solve regression problems in an efficient and stable way. The QR factorization can also help identify the most "important" or "representative" columns of the data set, an area of matrix theory known as *interpolatory decompositions*.



Theorem 3.5. *Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$ has linearly independent columns (hence $m \geq n$). Then there exists a subunitary matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$ and an invertible upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ such that*

$$\mathbf{A} = \mathbf{QR}.$$

EXERCISES

3.6. Suppose the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m$ are *orthonormal* to begin with. What can you say about $\mathbf{Q}$ and $\mathbf{R}$?

3.7. Suppose the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m$ are *orthogonal* but not necessarily *normalized* to be unit vectors. What can you say about the matrix $\mathbf{R}$? Be as specific as possible.

Example 3.1. To illustrate the QR factorization, let us *orthogonalize* the "bad basis" from page 23:

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \mathbf{a}_2 = \begin{bmatrix} 1 \\ .001 \end{bmatrix}.$$

The first step simply normalizes $\mathbf{a}_1$:

$$\widehat{\mathbf{q}}_1 := \mathbf{a}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad r_{1,1} := \|\widehat{\mathbf{q}}_1\| = 1, \qquad \mathbf{q}_1 := \frac{1}{r_{1,1}}\widehat{\mathbf{q}}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The second step begins by removing from $\mathbf{a}_2$ the best approximation from $\operatorname{span}\{\mathbf{q}_1\}$:

$$r_{1,2} := \mathbf{q}_1^T \mathbf{a}_2 = 1, \qquad \widehat{\mathbf{q}}_2 := \mathbf{a}_2 - r_{1,1}\mathbf{q}_1 = \begin{bmatrix} 1 \\ .001 \end{bmatrix} - 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ .001 \end{bmatrix}.$$

Complete the second step by normalizing:

$$r_{2,2} := \|\widehat{\mathbf{q}}_2\| = .001, \qquad \mathbf{q}_2 := \frac{1}{r_{2,2}}\widehat{\mathbf{q}}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Now assemble the pieces into the QR decomposition:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & .001 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & .001 \end{bmatrix} = \mathbf{QR}.$$

This example is not so exciting (after all, $\mathbf{Q} = \mathbf{I}$ in this special case), but it does illustrate how a *small diagonal entry $r_{j,j}$ in $\mathbf{R}$ indicates that one of the $\mathbf{a}_j$ vectors was nearly captured entirely by $\mathbf{q}_1, \ldots, \mathbf{q}_{j-1}$.*

A variation called the *rank-revealing QR decomposition* automatically reorders the columns of $\mathbf{A}$ so that the small entries of $\mathbf{R}$ congregate at the bottom right.

3.3.1 *Coda: Testing orthogonality*

Recall the calculation of $\mathbf{Q}^T\mathbf{Q}$ from page 24:

$$\mathbf{Q}^T\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^T\mathbf{q}_1 & \mathbf{q}_1^T\mathbf{q}_2 & \cdots & \mathbf{q}_1^T\mathbf{q}_n \\ \mathbf{q}_2^T\mathbf{q}_1 & \mathbf{q}_2^T\mathbf{q}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{q}_{n-1}^T\mathbf{q}_n \\ \mathbf{q}_n^T\mathbf{q}_1 & \cdots & \mathbf{q}_n^T\mathbf{q}_{n-1} & \mathbf{q}_n^T\mathbf{q}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} = \mathbf{I}.$$

Key concept. The (j,k) entry of the matrix $\mathbf{Q}^T\mathbf{Q}$ is the inner product of $\mathbf{q}_j$ and $\mathbf{q}_k$:

$$(\mathbf{Q}^T\mathbf{Q})_{j,k} = \mathbf{q}_j^T\mathbf{q}_k.$$

Thus, computing $\mathbf{Q}^T\mathbf{Q}$ gives a quick way to test the orthonormality of the columns of some matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$.

(i) If one of the *off-diagonal* entries of $\mathbf{Q}^T\mathbf{Q}$ is nonzero, then the columns are not orthogonal:

$$(\mathbf{Q}^T\mathbf{Q})_{j,k} \neq 0 \text{ for } j \neq k \quad \Longrightarrow \quad \mathbf{q}_j \text{ and } \mathbf{q}_k \text{ are not orthogonal.}$$

(ii) If one of the *diagonal* entries of $\mathbf{Q}^T\mathbf{Q}$ is not equal to one, then a column of $\mathbf{Q}$ is not normalized:

$$(\mathbf{Q}^T\mathbf{Q})_{j,j} \neq 1 \quad \Longrightarrow \quad \|\mathbf{q}_j\| \neq 1.$$

(iii) If $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, then the columns of $\mathbf{Q}$ are orthonormal.

When testing $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ on a computer, you will inevitably encounter very small rounding errors due to the machine's finite precision floating point arithmetic. In particular, it is common to find that $(\mathbf{Q}^T\mathbf{Q})_{j,k} \approx 10^{-15}$ for $j \neq k$. This is nothing to worry about: we can regard such vectors as orthogonal.

If you try to implement the Gram–Schmidt process (as described above) on a computer with moderate values of m and n, you will likely produce vectors that fail to be orthogonal by a non-trivial margin (e.g., $(\mathbf{Q}^T\mathbf{Q})_{j,k} \approx 10^{-8}$ or larger). A variant of the algorithm called the "Modified Gram–Schmidt" method produces better results; you can improve the $\mathbf{q}_j$ vectors even more using a "reorthogonalization" algorithm or the "Householder QR" method. You can learn about such topics in a course in Numerical Linear Algebra.

These considerations inform robust numerical software for QR decompositions, such as the LAPACK library that is utilized by NumPy.

Example 3.2. First consider

$$\mathbf{Q} = \begin{bmatrix} 1 & 1/\sqrt{3} \\ 0 & 1/\sqrt{3} \\ 0 & 1/\sqrt{3} \end{bmatrix}.$$

Then

$$\mathbf{Q}^T\mathbf{Q} = \begin{bmatrix} 1 & 1/\sqrt{3} \\ 1/\sqrt{3} & 1 \end{bmatrix}.$$

The off-diagonal entries of $\mathbf{Q}^T\mathbf{Q}$ are nonzero, so by (i) we conclude that the columns of $\mathbf{Q}$ are not orthogonal. The diagonal entries of $\mathbf{Q}^T\mathbf{Q}$ both equal 1, implying that the columns of $\mathbf{Q}$ are normalized (i.e., $\|\mathbf{q}_1\| = \|\mathbf{q}_2\| = 1$).

Next consider

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

Then

$$\mathbf{Q}^T\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}.$$

The off-diagonal entries of $\mathbf{Q}^T\mathbf{Q}$ are zero, so we conclude that the columns of $\mathbf{Q}$ are orthogonal. The diagonal entries of $\mathbf{Q}^T\mathbf{Q}$ do not equal 1, implying by (ii) that the columns of $\mathbf{Q}$ are not normalized: indeed $\|\mathbf{q}_1\|^2 = 2$ and $\|\mathbf{q}_2\|^2 = 3$.

Suppose we normalize $\mathbf{q}_1$ and $\mathbf{q}_2$ by dividing each one by their norm, giving

$$\mathbf{Q} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{3} \\ 0 & 1/\sqrt{3} \end{bmatrix}.$$

Now

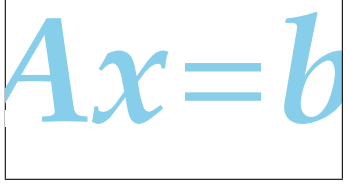
$$\mathbf{Q}^T\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

so by (iii) we conclude that the columns of $\mathbf{Q}$ are orthogonal.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 4
The Symmetric Eigenvalue Problem

Eigenvalues and eigenvectors are fundamental objects in linear algebra, especially enabling the analysis of dynamical systems and for identifying dominant features in data. We presume that students are already acquainted with computing eigenvalues and eigenvectors, but we will review a few key ideas.

4.1 Computing eigenvalues and eigenvectors

In this course we will mainly be interested in the eigenvalues of symmetric matrices, but we first start with the general case. This requires that we allow the possibility of *complex* eigenvalues $\lambda \in \mathbb{C}$ and eigenvectors $\mathbf{v} \in \mathbb{C}^n$.

Recall: $\mathbf{S}$ is symmetric if $\mathbf{S}^T = \mathbf{S}$, e.g.,

$$\mathbf{S} = \begin{bmatrix} a & b \\ b & d \end{bmatrix}.$$

Definition 4.1. *A scalar $\lambda \in \mathbb{C}$ is an* eigenvalue *of the square matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ provided there exists some nonzero vector $\mathbf{v} \in \mathbb{C}^n$, called an* eigenvector, *such that*

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v}.$$

If we allowed $\mathbf{v} = \mathbf{0}$, then any $\lambda \in \mathbb{C}$ would be an eigenvector: $\mathbf{S0} = \lambda\mathbf{0}$.

Subtract the left from the right-hand side to obtain

$$(\lambda\mathbf{I} - \mathbf{S})\mathbf{v} = \mathbf{0}.$$

This statement means that the null space $\mathcal{N}(\lambda\mathbf{I} - \mathbf{S})$ contains the nonzero vector $\mathbf{v} \in \mathbb{C}^n$, which means that $\lambda\mathbf{I} - \mathbf{S}$ *is not invertible*.

The null space $\mathcal{N}(\cdot)$ was defined on page 21. Apply that definition to $\lambda\mathbf{I} - \mathbf{S}$ (now allowing complex vectors):

$$\mathcal{N}(\lambda\mathbf{I} - \mathbf{S}) = \{\mathbf{v} \in \mathbb{C}^n : (\lambda\mathbf{I} - \mathbf{S})\mathbf{v} = \mathbf{0}\}.$$

Key concept. The scalar $\lambda \in \mathbb{C}$ is an eigenvalue of $\mathbf{S}$ if and only if $\lambda\mathbf{I} - \mathbf{S}$ is not invertible.

This key fact motivates our customary method for computing eigenvalues "by hand."

- A square matrix is not invertible if and only if its *determinant* is zero.
- Thus to find the values of $\lambda \in \mathbb{C}$ where $\lambda\mathbf{I} - \mathbf{S}$ is not invertible, compute the determinant of $\lambda\mathbf{I} - \mathbf{S}$, written $\det(\lambda\mathbf{I} - \mathbf{S})$.
- If $\mathbf{S} \in \mathbb{R}^{n\times n}$, then $\det(\lambda\mathbf{I} - \mathbf{S})$ is a degree-n polynomial, called the *characteristic polynomial*.
- Factor $\det(\lambda\mathbf{I} - \mathbf{S}) = 0$ to find the eigenvalues of $\mathbf{S}$.
- Once you have an eigenvalue λ of $\mathbf{S}$, find a corresponding eigenvector by solving $(\lambda\mathbf{I} - \mathbf{S})\mathbf{v} = \mathbf{0}$ for a *nonzero* solution $\mathbf{v} \in \mathbb{C}^n$.

Early computer algorithms that tried to implement this process for larger matrices were highly problematic. Modern algorithms compute eigenvalues using an entirely different approach: far more subtle and interesting. To learn more, read up on "the QR algorithm for eigenvalues."

A nonzero solution $\mathbf{v}$ must always exist, since $\lambda\mathbf{I} - \mathbf{S}$ is not invertible for this λ. Equivalently, you are looking for any nonzero $\mathbf{v} \in \mathcal{N}(\lambda\mathbf{I} - \mathbf{S})$.

Example 4.1. Consider the 2×2 matrix

$$\mathbf{S} = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}.$$

Compute the characteristic polynomial:

$$\det(\lambda\mathbf{I} - \mathbf{S}) = \det\left(\begin{bmatrix} \lambda - 2 & -2 \\ -2 & \lambda - 5 \end{bmatrix}\right)$$

$$= (\lambda - 2)(\lambda - 5) - (-2)\cdot(-2) = \lambda^2 - 7\lambda + 6.$$

Now factor $\lambda^2 - 7\lambda + 6 = 0$ to get

$$(\lambda - 1)(\lambda - 6) = 0,$$

and identify the two roots as the eigenvalues:

$$\lambda_1 = 1, \qquad \lambda_2 = 6.$$

Before computing eigenvectors, let us pause to recall where we are at.

Key concept. The scalar $\lambda \in \mathbb{C}$ is an eigenvalue of $\mathbf{S}$ if and only if $\lambda\mathbf{I} - \mathbf{S}$ is not invertible.

What would we get if we tried to compute $(\lambda\mathbf{I} - \mathbf{S})^{-1}$ directly, with λ as a variable? Gauss–Jordan elimination yields

$$(\lambda\mathbf{I} - \mathbf{S})^{-1} = \begin{bmatrix} \dfrac{\lambda - 5}{(\lambda - 1)(\lambda - 6)} & \dfrac{2}{(\lambda - 1)(\lambda - 6)} \\ \dfrac{2}{(\lambda - 1)(\lambda - 6)} & \dfrac{\lambda - 2}{(\lambda - 1)(\lambda - 6)} \end{bmatrix}. \tag{4.1}$$

The matrix $(\lambda\mathbf{I} - \mathbf{S})^{-1}$ is sufficiently important to have its own special name: we call it the *resolvent of* $\mathbf{S}$ *at* λ.

Take a moment to savor this formula! When is $\lambda\mathbf{I} - \mathbf{S}$ not invertible? Precisely when the formula for $(\lambda\mathbf{I} - \mathbf{S})^{-1}$ fails – that is, when

we get a *division by zero*. Notice that this happens only when the denominator of any entry in the matrix (4.1) is zero. In this case, the denominator is precisely

$$(\lambda - 1)(\lambda - 6) = \lambda^2 - 7\lambda + 6,$$

the characteristic polynomial! When $\lambda^2 - 7\lambda + 6 = 0$, λ is an eigenvalue of $\mathbf{S}$, and $(\lambda\mathbf{I} - \mathbf{S})^{-1}$ does not exist (due to the division by zero).

Now we return to the computation of the eigenvectors. To compute an eigenvector $\mathbf{v}_1$ corresponding to $\lambda_1 = 1$, we must find a *nontrivial* (nonzero) solution $\mathbf{v}_1$ to the system

$$(\lambda_1\mathbf{I} - \mathbf{S})\mathbf{v}_1 = \mathbf{0}.$$

In the language of null spaces introduced at the beginning of this chapter, we seek a *nonzero* vector $\mathbf{v}_1 \in \mathcal{N}(\lambda_1\mathbf{I} - \mathbf{S})$.

Writing out the matrix explicitly, we need to solve

$$(\lambda_1\mathbf{I} - \mathbf{S})\mathbf{v}_1 = \begin{bmatrix} -1 & -2 \\ -2 & -4 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

for α and β not both zero. The matrix equation gives two scalar equations,

$$-\alpha - 2\beta = 0, \qquad -2\alpha - 4\beta = 0,$$

which both reduce to $\alpha = -2\beta$. Thus $\mathbf{v}_1$ can be any vector of the form

$$\mathbf{v}_1 = \begin{bmatrix} -2\beta \\ \beta \end{bmatrix}, \qquad \beta \neq 0.$$

Notice that these vectors (aside from the omission of $\mathbf{v}_1 = \mathbf{0}$) form a one-dimensional subspace, $\mathcal{V}_1 := \text{span}\{[-2, 1]^T\}$. The eigenvector identifies a *direction*. Any nonzero multiple of that direction is still an eigenvector associated with the same eigenvalue.

We will often prefer to pick $\mathbf{v}_1$ to be a unit vector. In this case, either of these two choices would work:

$$\mathbf{v}_1 = \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} \quad \text{or} \quad \mathbf{v}_1 = \begin{bmatrix} 2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}.$$

To compute the eigenvector associated with $\lambda_2 = 6$, we must find a nonzero solution of

$$(\lambda_2\mathbf{I} - \mathbf{S})\mathbf{v}_1 = \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

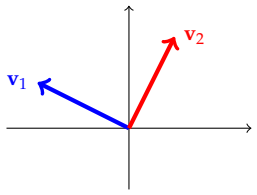
which reduces to the single equation $\beta = 2\alpha$ for $\alpha \neq 0$. Thus

$$\mathbf{v}_2 = \begin{bmatrix} \alpha \\ 2\alpha \end{bmatrix}, \qquad \alpha \neq 0,$$

and we could choose unit-length eigenvectors to be

$$\mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} \quad \text{or} \quad \mathbf{v}_2 = \begin{bmatrix} -1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix}.$$

Even better: since $\mathbf{v}_1$ is an *eigenvector*, we also know that $\mathbf{S}\mathbf{v}_1 = \lambda_1\mathbf{v}_1$ points in the same direction as $\mathbf{v}_1$: in this special direction $\mathbf{v}_1$, multiplying by the *matrix* $\mathbf{S}$ has the same effect as multiplying by the *scalar* λ_1.



Eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of the symmetric matrix $\mathbf{S}$. (Notice that these vectors are orthogonal.) Any nonzero vector in the direction of $\mathbf{v}_1$ is an eigenvector for λ_1; any nonzero vector in the direction of $\mathbf{v}_2$ is an eigenvector for λ_2.

4.2 Eigenvalues and eigenvectors of symmetric matrices

To derive the singular value decomposition of a general (rectangular) matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we shall rely on several special properties of the square, symmetric matrix $\mathbf{A}^T\mathbf{A}$. While this course assumes you are well acquainted with eigenvalues and eigenvectors, we will recall some fundamental concepts, especially pertaining to symmetric matrices.

4.2.1 A passing nod to complex numbers

Recall that even if a matrix has real number entries, it could have eigenvalues that are complex numbers; the corresponding eigenvectors will also have complex entries. Consider, for example, the matrix

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

To find the eigenvalues of $\mathbf{S}$, form the *characteristic polynomial*

$$\det(\lambda\mathbf{I} - \mathbf{S}) = \det\left(\begin{bmatrix} \lambda & 1 \\ -1 & \lambda \end{bmatrix}\right) = \lambda^2 + 1.$$

Factor this polynomial (e.g., using the quadratic formula) to get

$$\det(\lambda\mathbf{I} - \mathbf{S}) = \lambda^2 + 1 = (\lambda - \mathrm{i})(\lambda + \mathrm{i}),$$

where $\mathrm{i} = \sqrt{-1}$. Thus, we conclude that $\mathbf{S}$ (a matrix with *real* entries) has the *complex* eigenvalues

$$\lambda_1 = \mathrm{i}, \qquad \lambda_2 = -\mathrm{i}$$

and we can compute the corresponding eigenvectors

$$\mathbf{v}_1 = \begin{bmatrix} \mathrm{i} \\ 1 \end{bmatrix}, \qquad \mathbf{v}_2 = \begin{bmatrix} -\mathrm{i} \\ 1 \end{bmatrix}.$$

Suppose we want to compute the norm of the eigenvector $\mathbf{v}_1$. Using our usual method, we would have

$$\|\mathbf{v}_1\|^2 = \mathbf{v}_1^T\mathbf{v}_1 = \begin{bmatrix} \mathrm{i} & 1 \end{bmatrix}\begin{bmatrix} \mathrm{i} \\ 1 \end{bmatrix} = \mathrm{i}^2 + 1 = -1 + 1 = 0.$$

This result seems strange, no? How could the norm of a nonzero vector – even one with complex entries – be zero?

This example reveals a crucial shortcoming in our definition of the norm, when applied to complex vectors. Instead of

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\mathbf{x}^T\mathbf{x}},$$

To find the eigenvector associated with λ_1, we need to find some nonzero $\mathbf{v} \in \mathcal{N}(\lambda_1\mathbf{I} - \mathbf{S})$. To do so, solve the *consistent but underdetermined* system

$$\begin{bmatrix} \mathrm{i} & 1 \\ -1 & \mathrm{i} \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The first row requires

$$\mathrm{i}\alpha + \beta = 0,$$

while the second row requires

$$-\alpha + \mathrm{i}\beta = 0.$$

Multiply that last equation by $-\mathrm{i}$ and you obtain the first equation: so if you satisfy the second equation ($\alpha = \mathrm{i}\beta$), you satisfy them both. Thus let

$$\mathbf{v} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \mathrm{i}\beta \\ \beta \end{bmatrix}.$$

The specific eigenvector $\mathbf{v}_1$ presented in the main text follows from picking $\beta = 1$.

we want

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

For real vectors $\mathbf{x} \in \mathbb{R}^n$, both definitions are the same. For complex vectors $\mathbf{x} \in \mathbb{C}^n$ they can be very different. Just as the norm of a real vector has the compact notation $\|\mathbf{v}\| = \sqrt{\mathbf{v}^T\mathbf{v}}$, so too does the norm of a complex vector:

$$\|\mathbf{v}\| = \sqrt{\overline{\mathbf{v}}^T\mathbf{v}},$$

where $\overline{\mathbf{v}}$ denotes the *complex conjugate* of $\mathbf{v}$. Now apply this definition of the norm to $\mathbf{v}_1$:

$$\|\mathbf{v}_1\|^2 = \overline{\mathbf{v}}_1^T\mathbf{v}_1 = \begin{bmatrix} -\mathrm{i} & 1 \end{bmatrix} \begin{bmatrix} \mathrm{i} \\ 1 \end{bmatrix} = -\mathrm{i}^2 + 1 = 1 + 1 = 2,$$

a much more reasonable answer than we had before.

We will wrap up this complex interlude by proving that the real symmetric matrices that will be our focus in this course can never have complex eigenvalues.

4.2.2 *The spectral theorem for symmetric matrices*

Theorem 4.1. *All eigenvalues of a real symmetric matrix are real.*

Proof. Let $\mathbf{S}$ denote a symmetric matrix with real entries, so $\mathbf{S}^T = \mathbf{S}$ (since $\mathbf{S}$ is symmetric) and $\overline{\mathbf{S}} = \mathbf{S}$ (since $\mathbf{S}$ is real).

Let $(\lambda, \mathbf{v})$ be an arbitrary eigenpair of $\mathbf{S}$, so that $\mathbf{S}\mathbf{v} = \lambda\mathbf{v}$. Without loss of generality, we can assume that $\mathbf{v}$ is scaled so that $\|\mathbf{v}\| = 1$, i.e., $\overline{\mathbf{v}}^T\mathbf{v} = \|\mathbf{v}\|^2 = 1$. Thus

$$\lambda = \lambda\|\mathbf{v}\|^2 = \lambda(\overline{\mathbf{v}}^T\mathbf{v}) = \overline{\mathbf{v}}^T(\lambda\mathbf{v}) = \overline{\mathbf{v}}^T(\mathbf{S}\mathbf{v}).$$

Since $\mathbf{S}$ is real and symmetric, $\mathbf{S} = \overline{\mathbf{S}}^T$, and so

$$\overline{\mathbf{v}}^T(\mathbf{S}\mathbf{v}) = \overline{\mathbf{v}}^T\overline{\mathbf{S}}^T\mathbf{v} = \overline{(\mathbf{S}\mathbf{v})}^T\mathbf{v} = \overline{(\lambda\mathbf{v})}^T\mathbf{v} = \overline{\lambda}\overline{\mathbf{v}}^T\mathbf{v} = \overline{\lambda}\|\mathbf{v}\|^2 = \overline{\lambda}.$$

We have shown that $\lambda = \overline{\lambda}$, which is only possible if λ is real. ∎

It immediately follows that if λ is an eigenvalue of the real symmetric matrix $\mathbf{S}$, then we can always find a real-valued eigenvector $\mathbf{v}$ of $\mathbf{S}$ corresponding to λ, simply by finding a real-valued vector in the null space

$$\mathcal{N}(\lambda\mathbf{I} - \mathbf{S}),$$

since $\lambda\mathbf{I} - \mathbf{S}$ is a real-valued matrix.

Crucially, the eigenvectors of a real symmetric matrix $\mathbf{S}$ associated with distinct eigenvalues must be orthogonal.

Theorem 4.2. *Eigenvectors of a real symmetric matrix associated with distinct eigenvalues are orthogonal.*

If $z = a + \mathrm{i}b \in \mathbb{C}$ with $a, b \in \mathbb{R}$, then $|z| = \sqrt{a^2 + b^2}$. We call $|z|$ then *magnitude* of the complex number z.

The *complex conjugate* of $z = a + \mathrm{i}b$ is

$$\overline{z} = a - \mathrm{i}b,$$

allowing us to write

$$\begin{aligned} |z|^2 = \overline{z}z &= (a - \mathrm{i}b)(a + \mathrm{i}) \\ &= a^2 - \mathrm{i}b + \mathrm{i}b + a^2 = a^2 + b^2. \end{aligned}$$

The complex conjugate of a matrix or vector applies *entrywise*. For example, if

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

then

$$\overline{\mathbf{x}} = \begin{bmatrix} \overline{x}_1 \\ \vdots \\ \overline{x}_n \end{bmatrix}, \qquad \overline{\mathbf{x}}^T = \begin{bmatrix} \overline{x}_1 & \cdots & \overline{x}_n \end{bmatrix}.$$

As a result, we can swap the order of conjugation and transposition:

$$(\overline{\mathbf{x}})^T = \overline{\mathbf{x}^T}.$$

Similarly, we can distribute conjugation over matrix-vector products:

$$\overline{(\mathbf{A}\mathbf{x})} = (\overline{\mathbf{A}})(\overline{\mathbf{x}}).$$

Since we do not yet know that $\mathbf{v}$ is real-valued, we must use the norm definition for complex vectors discussed in the previous subsection.

If $z = a + \mathrm{i}b$ and $z = \overline{z}$, then $a + \mathrm{i}b = a - \mathrm{i}b$, i.e.,

$$b = -b,$$

which is only possible if $b = 0$.

Proof. Suppose λ and γ are distinct eigenvalues of a real symmetric matrix $\mathbf{S}$ associated with eigenvectors $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$:

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v}, \qquad \mathbf{S}\mathbf{w} = \gamma\mathbf{w}$$

with $\lambda \neq \gamma$. Now consider

$$\lambda\mathbf{w}^T\mathbf{v} = \mathbf{w}^T(\lambda\mathbf{v}) = \mathbf{w}^T(\mathbf{S}\mathbf{v}) = \mathbf{w}^T\mathbf{S}^T\mathbf{v},$$

where we have used the fact that $\mathbf{S} = \mathbf{S}^T$. Now

$$\mathbf{w}^T\mathbf{S}^T\mathbf{v} = (\mathbf{S}\mathbf{w})^T\mathbf{v} = (\gamma\mathbf{w})^T\mathbf{v} = \gamma\mathbf{w}^T\mathbf{v}.$$

We have thus shown that

$$\lambda\mathbf{w}^T\mathbf{v} = \gamma\mathbf{w}^T\mathbf{v}.$$

Since $\lambda \neq \gamma$, this statement can only be true if $\mathbf{w}^T\mathbf{v} = 0$, i.e., if $\mathbf{v}$ and $\mathbf{w}$ are orthogonal. ∎

What if the eigenvalues are not *distinct*? Consider the simple 2×2 identity matrix, $\mathbf{I}$. Any nonzero $\mathbf{x} \in \mathbb{R}^2$ is an eigenvector of $\mathbf{I}$ associated with the eigenvalue $\lambda = 1$, since

$$\mathbf{I}\mathbf{x} = 1\mathbf{x}.$$

Thus we have many eigenvectors that are not orthogonal. However, we can always find vectors, like

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

that are orthogonal.

We are ready to collect relevant facts in the *Spectral Theorem*.

Theorem 4.3 (Spectral Theorem)**.** *Suppose $\mathbf{S} \in \mathbb{R}^{n\times n}$ is symmetric, $\mathbf{S}^T = \mathbf{S}$. Then there exist n (not necessarily distinct) eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding unit-length eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ such that*

$$\mathbf{S}\mathbf{v}_j = \lambda_j\mathbf{v}_j, \tag{4.2}$$

the eigenvectors form an orthonormal *basis for $\mathbb{R}^n$,*

$$\mathbb{R}^n = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_n\},$$

and $\mathbf{v}_j^T\mathbf{v}_k = 0$ when $j \neq k$, and $\mathbf{v}_j^T\mathbf{v}_j = \|\mathbf{v}_j\|^2 = 1$.

4.2.3 *Two important ways to write the spectral decomposition*

The Spectral Theorem leads to two very convenient ways to decompose the matrix $\mathbf{S}$. First stack the n equations in (4.2) side-by-side:

$$\begin{bmatrix} | & | & & | \\ \mathbf{S}\mathbf{v}_1 & \mathbf{S}\mathbf{v}_2 & \cdots & \mathbf{S}\mathbf{v}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \lambda_1\mathbf{v}_1 & \lambda_2\mathbf{v}_2 & \cdots & \lambda_n\mathbf{v}_n \\ | & | & & | \end{bmatrix}.$$

The matrices on each side of this equation can be pulled apart as the product of two simpler matrices:

$$\mathbf{S}\begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}.$$

For example, when

$$\mathbf{S} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix},$$

we have $\lambda_1 = 4$ and $\lambda_2 = 2$, with

$$\mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}.$$

Note that these eigenvectors are unit vectors, and they are orthogonal. We can write

$$\begin{aligned} \mathbf{S} &= \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \\ &= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \\ &= \lambda_1\mathbf{v}_1\mathbf{v}_1^T + +\lambda_2\mathbf{v}_2\mathbf{v}_2^T \\ &= 4\begin{bmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{bmatrix} + 2\begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}. \end{aligned}$$

Define the matrices

$$\mathbf{V} := \begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix}, \qquad \mathbf{\Lambda} := \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix},$$

so the last equation can be written compactly as

$$\mathbf{SV} = \mathbf{V\Lambda}. \tag{4.3}$$

Since the vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ for an *orthonormal* basis for $\mathbb{R}^n$, notice that this orthogonality can be neatly summarized as:

$$\begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix}^T \begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^T\mathbf{v}_1 & \mathbf{v}_1^T\mathbf{v}_2 & \cdots & \mathbf{v}_1^T\mathbf{v}_n \\ \mathbf{v}_2^T\mathbf{v}_1 & \mathbf{v}_2^T\mathbf{v}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{v}_{n-1}^T\mathbf{v}_n \\ \mathbf{v}_n^T\mathbf{v}_1 & \cdots & \mathbf{v}_n^T\mathbf{v}_{n-1} & \mathbf{v}_n^T\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} = \mathbf{I}.$$

We have just seen that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. Since $\mathbf{V} \in \mathbb{R}^{n\times n}$ is a square matrix, this must mean that $\mathbf{V}^{-1} = \mathbf{V}^T$. Multiplying equation (4.3) on the right by $\mathbf{V}^T$ gives

$$\mathbf{S} = \mathbf{V\Lambda V}^T \tag{4.4}$$

$\mathbf{SVV}^T = \mathbf{SI} = \mathbf{S}$.

Equation (4.4) is called the *diagonalization of the symmetric matrix* **S**.

We can also rearrange the decomposition (4.4) in a different format that further illuminates the structure and action of **S**. Start with (4.4) and multiply together $\mathbf{V\Lambda}$, giving

$$\mathbf{S} = (\mathbf{V\Lambda})\mathbf{V}^T = \begin{bmatrix} | & | & & | \\ \lambda_1\mathbf{v}_1 & \lambda_2\mathbf{v}_2 & \cdots & \lambda_n\mathbf{v}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} — & \mathbf{v}_1^T & — \\ — & \mathbf{v}_2^T & — \\ & \vdots & \\ — & \mathbf{v}_n^T & — \end{bmatrix}.$$

If this looks like magic, a small symbolic calculation will convince you that this works. Multiply out

$$\begin{bmatrix} a & e \\ b & f \\ c & g \\ d & h \end{bmatrix} \begin{bmatrix} s & t & u & v \\ w & x & y & z \end{bmatrix}$$

and compare the result to

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \begin{bmatrix} s & t & u & v \end{bmatrix} + \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} \begin{bmatrix} w & x & y & z \end{bmatrix}.$$

Notice that the matrices on the right form a *matrix outer product*. Multiplying them together, row-by-column, reveals that entries from $\lambda_1\mathbf{v}_1$ always multiply against entries from $\mathbf{v}_1^T$, and $\lambda_2\mathbf{v}_2$ multiplies against $\mathbf{v}_2^T$, and so on. We can thus write

$$\mathbf{S} = \sum_{j=1}^{n} \lambda_j \begin{bmatrix} | \\ \lambda_1\mathbf{v}_j \\ | \end{bmatrix} \begin{bmatrix} — & \mathbf{v}_j^T & — \end{bmatrix} = \sum_{j=1}^{n} \lambda_j \begin{bmatrix} & \\ \mathbf{v}_j\mathbf{v}_j^T & \\ & \end{bmatrix}.$$

$$\mathbf{S} = \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \mathbf{v}_j^T. \tag{4.5}$$

This equation expresses $\mathbf{S}$ as the weighted sum of the special n-by-n matrices $\mathbf{v}_j \mathbf{v}_j^T$. Notice that $\mathbf{v}_j \mathbf{v}_j^T$ is the *orthogonal projector onto* $\operatorname{span}\{\mathbf{v}_j\}$, the direction of the jth eigenvector. This perspective gives a beautiful view of the action of $\mathbf{S}$ upon a vector $\mathbf{x} \in \mathbb{R}^n$:

In the next chapter, the singular value decomposition will provide a similar way to tease apart a rectangular matrix.

$$\mathbf{S}\mathbf{x} = \left(\sum_{j=1}^{n} \lambda_j \mathbf{v}_j \mathbf{v}_j^T \right) \mathbf{x} = \sum_{j=1}^{n} \lambda_j \left(\mathbf{v}_j \mathbf{v}_j^T \mathbf{x} \right) = \sum_{j=1}^{n} \lambda_j (\mathbf{v}_j^T \mathbf{x}) \mathbf{v}_j. \tag{4.6}$$

The vector $\mathbf{S}\mathbf{x}$ is constructed by:

- Computing the best approximation to $\mathbf{x}$ from $\operatorname{span}\{\mathbf{v}_j\}$,

  $$\frac{\mathbf{v}_j \mathbf{v}_j^T}{\mathbf{v}_j^T \mathbf{v}_j} \mathbf{x} = \mathbf{v}_j \mathbf{v}_j^T \mathbf{x} = (\mathbf{v}_j^T \mathbf{x}) \mathbf{v}_j,$$

  where we have used the normality of the eigenvector ($\mathbf{v}_j^T \mathbf{v}_j = \|\mathbf{v}_j\|^2 = 1$) and the best approximation formula (2.1) on page 13;

- Weighting the best approximation in the $\mathbf{v}_j$ direction by the eigenvalue λ_j:

  $$\lambda_j (\mathbf{v}_j^T \mathbf{x}) \mathbf{v}_j;$$

- Adding up the pieces to get $\mathbf{S}\mathbf{x}$.

The result will depend on two key factors: (1) how much $\mathbf{x}$ is biased toward each eigenvector direction (as revealed by $\mathbf{v}_j^T \mathbf{x}$), and (2) the size of each eigenvalue, λ_j.

EXERCISES

4.8. Let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be a symmetric matrix.

- For any positive integer k, show that

  $$\mathbf{S}^k = \sum_{j=1}^{n} \lambda_j^k \mathbf{v}_j \mathbf{v}_j^T.$$

- Show that if $\mathbf{S}$ is invertible,

  $$\mathbf{S}^{-1} = \sum_{j=1}^{n} \frac{1}{\lambda_j} \mathbf{v}_j \mathbf{v}_j^T.$$

  What goes wrong with this formula when $\mathbf{S}$ is not invertible?

4.3 *Symmetric positive definite matrices*

Amongst the symmetric matrices, we pick out a special class that arises in many applications: the *positive (semi)definite matrices*. Key to this definition is the *quadratic form*

For example, in statistics, covariance matrices are always symmetric positive semidefinite.

$$\mathbf{x}^T\mathbf{S}\mathbf{x}.$$

You could work this out in terms of the entries $s_{j,k}$ of $\mathbf{S}$ and the entries $\mathbf{x}_j$ of $\mathbf{x}$ to get

$$\mathbf{x}^T\mathbf{S}\mathbf{x} = \sum_{j=1}^{n}\sum_{k=1}^{n} s_{j,k}x_j x_k, \tag{4.7}$$

but this formula does not reveal very much. For example, can you use (4.7) to determine the maximum and minimum values of $\mathbf{x}^T\mathbf{S}\mathbf{x}$, over all unit vectors $\mathbf{x}$?

Eigenvalues provide much more insight. Using equation (4.6),

$$\begin{aligned}\mathbf{x}^T\mathbf{S}\mathbf{x} = \mathbf{x}^T(\mathbf{S}\mathbf{x}) &= \mathbf{x}^T\left(\sum_{j=1}^{n}\lambda_j(\mathbf{v}_j^T\mathbf{x})\mathbf{v}_j\right)\\ &= \sum_{j=1}^{n}\lambda_j\mathbf{x}^T(\mathbf{v}_j^T\mathbf{x})\mathbf{v}_j \;=\; \sum_{j=1}^{n}\lambda_j(\mathbf{v}_j^T\mathbf{x})(\mathbf{x}^T\mathbf{v}_j) \;=\; \sum_{j=1}^{n}\lambda_j(\mathbf{v}_j^T\mathbf{x})^2.\end{aligned}$$

This expression reveals the extreme values of $\mathbf{x}^T\mathbf{S}\mathbf{x}$ when $\mathbf{x}$ is required to be a unit vector. Suppose the eigenvalues are labeled in decreasing order (always possible, since the are real numbers):

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.$$

First consider the maximum value. Since λ_1 is the largest eigenvalue,

$$\mathbf{x}^T\mathbf{S}\mathbf{x} = \sum_{j=1}^{n}\lambda_j(\mathbf{v}_j^T\mathbf{x})^2 \leq \lambda_1\sum_{j=1}^{n}(\mathbf{v}_j^T\mathbf{x})^2. \tag{4.8}$$

As noted before, since $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, we also have $\mathbf{V}\mathbf{V}^T = \mathbf{I}$, and so

We can only say that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ implies $\mathbf{V}\mathbf{V}^T = \mathbf{I}$ because $\mathbf{V}$ is a square matrix.

$$\sum_{j=1}^{n}(\mathbf{v}_j^T\mathbf{x})^2 = \|\mathbf{V}^T\mathbf{x}\|^2 = (\mathbf{V}^T\mathbf{x})^T(\mathbf{V}^T\mathbf{x}) = \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|^2.$$

Thus, the inequality (4.8) becomes

$$\mathbf{x}^T\mathbf{S}\mathbf{x} \;\leq\; \lambda_1\sum_{j=1}^{n}(\mathbf{v}_j^T\mathbf{x})^2 \;=\; \lambda_1\|\mathbf{x}\|^2.$$

Maximizing $\mathbf{x}^T\mathbf{S}\mathbf{x}$ over all unit vectors thus gives

$$\max_{\|\mathbf{x}\|=1}\mathbf{x}^T\mathbf{S}\mathbf{x} \leq \lambda_1.$$

Choosing $\mathbf{x} = \mathbf{v}_1$ shows that this inequality holds with equality, since

$$\mathbf{v}_1^T \mathbf{S}\mathbf{v}_1 = \mathbf{v}_1^T(\mathbf{S}\mathbf{v}_1) = \mathbf{v}_1^T(\lambda_1 \mathbf{v}_1) = \lambda_1 \mathbf{v}_1^T \mathbf{v}_1 = \lambda_1.$$

You can readily see how to modify this argument to minimize $\mathbf{x}^T\mathbf{S}\mathbf{x}$.

Theorem 4.4. *Let $\mathbf{S} \in \mathbb{R}^{n\times n}$ be a symmetric matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Then*

$$\max_{\|\mathbf{x}\|=1} \mathbf{x}^T\mathbf{S}\mathbf{x} = \lambda_1, \qquad \min_{\|\mathbf{x}\|=1} \mathbf{x}^T\mathbf{S}\mathbf{x} = \lambda_n.$$

Notice a corollary of Theorem 4.4: For any unit vector $\mathbf{x} \in \mathbb{R}^n$,

$$\lambda_n \leq \mathbf{x}^T\mathbf{S}\mathbf{x} \leq \lambda_1.$$

(This result can be extended further into the Courant–Fischer variational characterization of eigenvalues.)

Matrices for which $\mathbf{x}^T\mathbf{S}\mathbf{x}$ is always positive (or nonnegative) arise in many applications, and so we given them a special name.

Definition 4.2. *A symmetric matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ is*

- positive definite *provided $\mathbf{x}^T\mathbf{S}\mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^n$;*
- positive semidefinite *provided $\mathbf{x}^T\mathbf{S}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$.*

For the example above,

$$\begin{aligned}\mathbf{x}^T\mathbf{S}\mathbf{x} &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 3x_1^2 - 2x_1x_2 + x_2^2 \\ &= 2(x_1 - x_2)^2 + (x_1 + x_2)^2.\end{aligned}$$

This last expression, the sum of squares, is clearly positive for all nonzero $\mathbf{x}$, so $\mathbf{S}$ is positive definite. (Proving positive definiteness by factoring polynomials like this would not be pleasant for larger values of n, hence the utility of Theorem 4.5!)

Theorem 4.5. *All eigenvalues of a symmetric positive definite matrix are positive; all eigenvalues of a symmetric positive semidefinite matrix are nonnegative.*

Proof. Let $(\lambda_j, \mathbf{v}_j)$ denote an eigenpair of the symmetric positive definite matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ with $\|\mathbf{v}_j\|^2 = \mathbf{v}_j^T\mathbf{v}_j = 1$. Since $\mathbf{S}$ is symmetric , λ_j must be real. We conclude that

$$\lambda_j = \lambda_j \mathbf{v}_j^T\mathbf{v}_j = \mathbf{v}_j^T(\lambda_j\mathbf{v}_j) = \mathbf{v}_j^T\mathbf{S}\mathbf{v}_j,$$

which must be positive since $\mathbf{S}$ is positive definite and $\mathbf{v}_j \neq \mathbf{0}$.

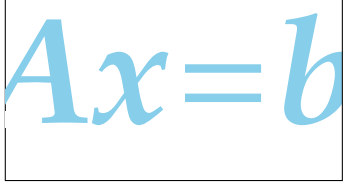
The proof for positive semidefinite matrices is the same, except we can only conclude that $\lambda_j = \mathbf{v}_j^T\mathbf{S}\mathbf{v}_j \geq 0$. ∎

Can you prove the converse of this theorem? (A symmetric matrix with positive eigenvalues is positive definite.) Hint: use the Spectral Theorem. With this result, we can check if $\mathbf{S}$ is positive definite by just looking at its eigenvalues, rather than working out a formula for $\mathbf{x}^T\mathbf{S}\mathbf{x}$, as done above.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 5
The Singular Value Decomposition

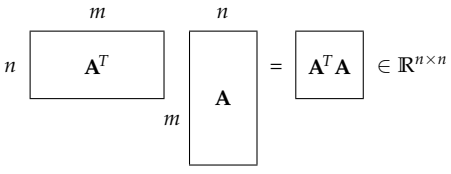
The singular value decomposition (SVD) is among the most important matrix factorizations, especially for applications in data science. Up to now, we have considered the range (column space) $\mathcal{R}(\mathbf{A})$ of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ in its entirety; the SVD will provide a way to *rank the directions in* $\mathcal{R}(\mathbf{A})$, allowing an analyst to pull out the most significant components from $\mathbf{A}$, while neglecting those that matter less for the application at hand. As we shall see, the SVD unlocks the solution to a variety of *least squares/regression problems*, and is also the fundamental tool for *dimension reduction* (including principal component analysis, PCA). The SVD is not just a practical tool; it also enables deep theoretical insights. This chapter introduces the SVD and describes a few of its most compelling applications.

5.1 Derivation of the singular value decomposition: Full rank case

We seek to derive the singular value decomposition of a general rectangular matrix. To simplify our initial derivation, we shall assume that $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$, and *all the columns of* $\mathbf{A}$ *linearly independent*. This last condition implies that $\operatorname{rank}(\mathbf{A})$ is as large as possible,

$$\operatorname{rank}(\mathbf{A}) = n,$$

so the dimension of the column space is n: $\dim(\mathcal{R}(\mathbf{A})) = n$.



To begin, form $\mathbf{A}^T\mathbf{A}$, which is an $n \times n$ matrix. Notice that $\mathbf{A}^T\mathbf{A}$ is always symmetric, since

$$(\mathbf{A}^T\mathbf{A})^T = \mathbf{A}^T(\mathbf{A}^T)^T = \mathbf{A}^T\mathbf{A}.$$

Furthermore, $\mathbf{A}^T\mathbf{A}$ is positive definite: notice that

$$\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} = (\mathbf{A}\mathbf{x})^T(\mathbf{A}\mathbf{x}) = \|\mathbf{A}\mathbf{x}\|^2 \geq 0.$$

Now suppose that $\mathbf{Ax} = \mathbf{0}$ for $\mathbf{x} \neq \mathbf{0}$. Remember that $\mathbf{Ax}$ is a linear combination of the columns of $\mathbf{A}$, so

$$\mathbf{0} = \mathbf{Ax} = x_1\mathbf{a}_1 + \cdots + x_n\mathbf{a}_n.$$

Since we assumed that the columns of $\mathbf{A}$ are linearly independent, the only way for $\mathbf{Ax} = \mathbf{0}$ is the trivial case, $\mathbf{x} = \mathbf{0}$. We conclude that

$$\mathbf{x}^T\mathbf{A}^T\mathbf{Ax} > 0 \text{ for all nonzero } \mathbf{x} \in \mathbb{R}^n,$$

and hence $\mathbf{A}^T\mathbf{A}$ is *symmetric positive definite*.

See Definition 4.2.

We are now ready to construct our first version of the singular value decomposition. We shall construct the pieces one at a time, then assemble them into the desired decomposition.

Step 1. **Compute the eigenvalues and eigenvectors of $\mathbf{A}^T\mathbf{A}$.**

As a consequence of results about symmetric matrices presented in Chapter 4, we can find n eigenpairs $\{(\lambda_j, \mathbf{v}_j)\}_{j=1}^n$ of $\mathbf{S} := \mathbf{A}^T\mathbf{A}$ with unit eigenvectors ($\mathbf{v}_j^T\mathbf{v}_j = \|\mathbf{v}_j\|^2 = 1$) that are orthogonal to one another ($\mathbf{v}_j^T\mathbf{v}_k = 0$ when $j \neq k$). We are free to pick any convenient indexing for these eigenpairs; we shall label them so that the eigenvalues are decreasing in size, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$. We emphasize that $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^n$. These orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are the *right singular vectors of* $\mathbf{A}$.

Even if $\mathbf{A}$ is a square matrix, be sure to compute the eigenvalues and eigenvectors of $\mathbf{A}^T\mathbf{A}$.

Since $\mathbf{S} = \mathbf{A}^T\mathbf{A}$ is positive definite, all its eigenvalues are positive.

Step 2. **Define $\sigma_j = \|\mathbf{Av}_j\| = \sqrt{\lambda_j}$ for $j = 1, \ldots, n$.**

Note that $\sigma_j^2 = \|\mathbf{Av}_j\|_2^2 = \mathbf{v}_j^T\mathbf{A}^T\mathbf{Av}_j = \lambda_j$. Since the eigenvalues $\lambda_1, \ldots, \lambda_n$ are decreasing in size, so too are the σ_j values:

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

The numbers $\sigma_1, \ldots, \sigma_n$ are called the *singular values of* $\mathbf{A}$.

Step 3. **Define $\mathbf{u}_j = \mathbf{Av}_j/\sigma_j$ for $j = 1, \ldots, n$.**

The assumption that $\text{rank}(\mathbf{A}) = n$ helped us out here, by ensuring that $\sigma_j > 0$ for all j: hence we can safely divide by σ_j in the definition of $\mathbf{u}_j$.

Notice that $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^m$. Because $\sigma_j = \|\mathbf{Av}_j\|$, we ensure that

$$\|\mathbf{u}_j\| = \left\|\frac{1}{\sigma_j}\mathbf{Av}_j\right\| = \frac{\|\mathbf{Av}_j\|}{\sigma_j} = 1.$$

Thus the $\mathbf{u}_j$ vectors are *automatically normalized*, by construction. Furthermore, these $\mathbf{u}_j$ vectors are orthogonal. To see this, write

$$\mathbf{u}_j^T\mathbf{u}_k = \frac{1}{\sigma_j\sigma_k}(\mathbf{Av}_j)^T(\mathbf{Av}_k) = \frac{1}{\sigma_j\sigma_k}\mathbf{v}_j^T\mathbf{A}^T\mathbf{Av}_k.$$

Since $\mathbf{v}_k$ is an eigenvector of $\mathbf{A}^T\mathbf{A}$ corresponding to eigenvalue λ_k,

$$\mathbf{u}_j^T\mathbf{u}_k = \frac{1}{\sigma_j\sigma_k}\mathbf{v}_j^T(\mathbf{A}^T\mathbf{A})\mathbf{v}_k = \frac{1}{\sigma_j\sigma_k}\mathbf{v}_j^T(\lambda_k\mathbf{v}_k) = \frac{\lambda_j}{\sigma_j\sigma_k}\mathbf{v}_j^T\mathbf{v}_k.$$

Since the eigenvectors of the symmetric matrix $\mathbf{A}^T\mathbf{A}$ are orthogonal, $\mathbf{v}_j^T\mathbf{v}_k = 0$ when $j \neq k$, so the $\mathbf{u}_j$ vectors inherit the orthogonality of the $\mathbf{v}_j$ vectors:

$$\mathbf{u}_j^T\mathbf{u}_k = 0, \qquad j \neq k.$$

These orthonormal vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$ are the *left singular vectors of* $\mathbf{A}$.

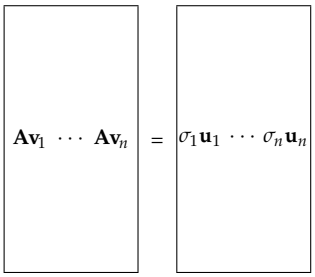
Step 4. **Put the pieces together.**

For all $j = 1, \ldots, n$,

$$\mathbf{A}\mathbf{v}_j = \sigma_j\mathbf{u}_j.$$

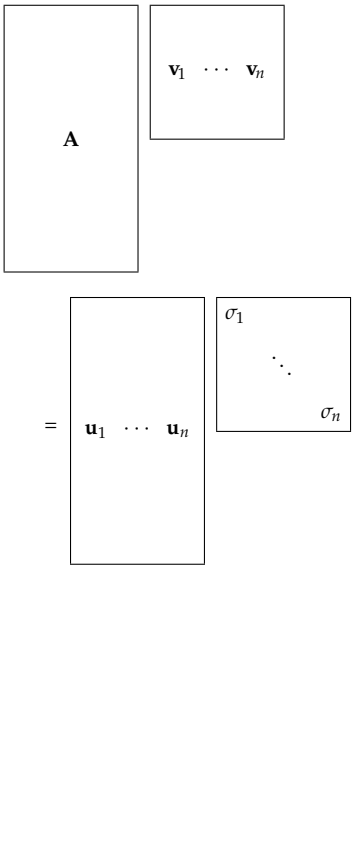
We can stack these n vector equations as columns of a single matrix equation:

$$\begin{bmatrix} | & | & & | \\ \mathbf{A}\mathbf{v}_1 & \mathbf{A}\mathbf{v}_2 & \cdots & \mathbf{A}\mathbf{v}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma_1\mathbf{u}_1 & \sigma_2\mathbf{u}_2 & \cdots & \sigma_n\mathbf{u}_n \\ | & | & & | \end{bmatrix}.$$



We follow the same strategy we used for the symmetric eigenvalue problem in Section 4.2.3. Note that both matrices in this equation can be factored into the product of simpler matrices:

$$\mathbf{A}\begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{bmatrix}\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}.$$



Denote these matrices as

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}, \tag{5.1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times n}$, and $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$.

We now have the essential ingredients for the two most important forms of the singular value decomposition. Since the eigenvectors $\mathbf{v}_j$ of the symmetric matrix $\mathbf{A}^T\mathbf{A}$ are orthonormal, the square matrix $\mathbf{V}$ has orthonormal columns. Just as in Chapter 4, this means that

$$\mathbf{V}^T\mathbf{V} = \begin{bmatrix} \mathbf{v}_1^T\mathbf{v}_1 & \mathbf{v}_1^T\mathbf{v}_2 & \cdots & \mathbf{v}_1^T\mathbf{v}_n \\ \mathbf{v}_2^T\mathbf{v}_1 & \mathbf{v}_2^T\mathbf{v}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{v}_{n-1}^T\mathbf{v}_n \\ \mathbf{v}_n^T\mathbf{v}_1 & \cdots & \mathbf{v}_n^T\mathbf{v}_{n-1} & \mathbf{v}_n^T\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} = \mathbf{I},$$

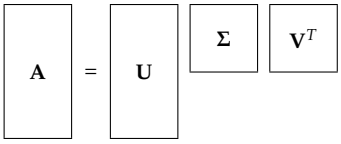
since the (j,k) entry of $\mathbf{V}^T\mathbf{V}$ is simply $\mathbf{v}_j^T\mathbf{v}_k$. Since $\mathbf{V}$ is square, the equation $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ implies that $\mathbf{V}^T = \mathbf{V}^{-1}$. Thus, in addition to $\mathbf{V}^T\mathbf{V}$, we also have

$$\mathbf{V}\mathbf{V}^T = \mathbf{V}\mathbf{V}^{-1} = \mathbf{I}.$$

The inverse of a square matrix is unique: since $\mathbf{V}^T$ does what the inverse of $\mathbf{V}$ is supposed to do, i.e., $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, it must be the unique matrix $\mathbf{V}^{-1}$.

Thus multiplying both sides of equation (5.1) on the right by $\mathbf{V}^T$ gives

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T. \tag{5.2}$$



This factorization is the *reduced* (or *economy-sized* or *skinny*) *singular value decomposition* of $\mathbf{A}$. It can be obtained via the Python/NumPy via the command

```
U, S, Vt = np.linalg.svd(A, FullMatrices=False),
```

Note that the third returned variable comes "pre-transposed," that is, the `np.linalg.svd` command returns the matrix `Vt` $= \mathbf{V}^T$. Since, in our case thus far, $\mathbf{V} \in \mathbb{R}^{n\times n}$ is a square matrix, one could easily miss this delicate point.

where `S` is a vector of singular values.

What can be said of the matrix $\mathbf{U} \in \mathbb{R}^{m\times n}$? Recall that its columns, the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$, are orthonormal. However, in contrast to $\mathbf{V}$, *we cannot conclude that* $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ *when* $m > n$. Why not? Because when $m > n$, $\mathbf{U}^T \in \mathbb{R}^{n\times m}$ has a nontrivial null space, and hence cannot be invertible.

When $m > n$, there must exist some nonzero $\mathbf{z} \in \mathbb{R}^m$ such that $\mathbf{z} \perp \mathbf{u}_1, \ldots, \mathbf{u}_n$, which implies $\mathbf{U}^T\mathbf{z} = \mathbf{0}$. Hence $\mathbf{U}\mathbf{U}^T\mathbf{z} = \mathbf{0}$, so we cannot have $\mathbf{U}\mathbf{U}^T = \mathbf{I}$. However, $\mathbf{U}\mathbf{U}^T \in \mathbb{R}^{m\times m}$ is a projector onto the n-dimensional subspace $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$ of $\mathbb{R}^m$.

Theorem 5.1 (Reduced Singular Value Decomposition, full-rank $\mathbf{A}$)**.** *Suppose* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *has* $\text{rank}(\mathbf{A}) = n$*, with* $m \geq n$*. Then we can write*

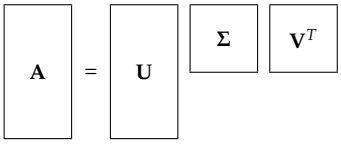
$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T,$$

where the columns of $\mathbf{U} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{V} \in \mathbb{R}^{n\times n}$ *are orthonormal,*

$$\mathbf{U}^T\mathbf{U} = \mathbf{I} \in \mathbb{R}^{n\times n}, \qquad \mathbf{V}^T\mathbf{V} = \mathbf{I} \in \mathbb{R}^{n\times n},$$

and $\Sigma \in \mathbb{R}^{n\times n}$ *is zero everywhere except for entries on the main diagonal, where the* (j,j) *entry is* σ_j*, for* $j = 1, \ldots, n$ *and*

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



Example 5.1. Consider the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ \sqrt{2} & -\sqrt{2} \end{bmatrix},$$

for which $\mathbf{A}^T\mathbf{A}$ is the symmetric matrix $\mathbf{S}$ that appeared earlier as an example on page 37:

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}.$$

This matrix has $\text{rank}(\mathbf{A}) = 2 = n$, so we can apply the analysis described above.

Step 1. **Compute the eigenvalues and eigenvectors of $\mathbf{A}^T\mathbf{A}$.**

We have already seen that, for this matrix, $\lambda_1 = 4$ and $\lambda_2 = 2$, with

$$\mathbf{v}_1 = \begin{bmatrix} \sqrt{2}/2 \\ -\sqrt{2}/2 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix},$$

with $\lambda_1 \geq \lambda_2$, the required order. The vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ will be the right singular vectors of $\mathbf{A}$.

Step 2. **Define $\sigma_j = \|\mathbf{A}\mathbf{v}_j\| = \sqrt{\lambda_j}$ for $j = 1, \ldots, n$.**

In this case, we compute

$$\sigma_1 = \sqrt{\lambda_1} = 2, \qquad \sigma_2 = \sqrt{\lambda_2} = \sqrt{2}.$$

Alternatively, we could have computed the singular values from

$$\mathbf{A}\mathbf{v}_1 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ \sqrt{2} & -\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

$$\mathbf{A}\mathbf{v}_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ \sqrt{2} & -\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 0 \\ 0 \end{bmatrix},$$

with $\sigma_1 = \|\mathbf{A}\mathbf{v}_1\| = 2$ and $\sigma_2 = \|\mathbf{A}\mathbf{v}_2\| = \sqrt{2}$.

Step 3. **Define $\mathbf{u}_j = \mathbf{A}\mathbf{v}_j/\sigma_j$, $j = 1, \ldots, n$.**

We use the vectors $\mathbf{A}\mathbf{v}_1$ and $\mathbf{A}\mathbf{v}_2$ computed at the last step:

$$\mathbf{u}_1 = \frac{1}{\sigma_1}\mathbf{A}\mathbf{v}_1 = \frac{1}{2}\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \qquad \mathbf{u}_2 = \frac{1}{\sigma_2}\mathbf{A}\mathbf{v}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} \sqrt{2} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Step 4. **Put the pieces together.**

We immediately have the *reduced SVD* $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$:

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

5.2 *The dyadic form of the SVD*

We are now prepared to develop an analogue of the formula (4.5) for full-rank rectangular matrices. Consider the reduced SVD,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

and multiply $\mathbf{U}\mathbf{\Sigma}$ to obtain

$$\begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma_1\mathbf{u}_1 & \sigma_1\mathbf{u}_2 & \cdots & \sigma_n\mathbf{u}_n \\ | & | & & | \end{bmatrix}.$$

Now notice that you can write $\mathbf{A} = (\mathbf{U}\boldsymbol{\Sigma})\mathbf{V}^T$ as

$$\begin{bmatrix} | & | & & | \\ \sigma_1\mathbf{u}_1 & \sigma_1\mathbf{u}_2 & \cdots & \sigma_n\mathbf{u}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} — & \mathbf{v}_1^T & — \\ — & \mathbf{v}_2^T & — \\ & \vdots & \\ — & \mathbf{v}_n^T & — \end{bmatrix} = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T,$$

which parallels the form (4.5) we had for symmetric matrices. For symmetric matrices $\mathbf{S} \in \mathbb{R}^{n\times n}$ we wrote

$$\mathbf{A} = \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \mathbf{v}_j^T.$$

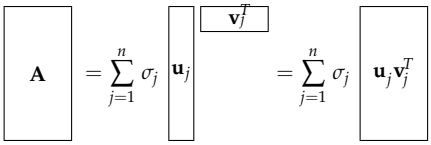
Now for any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ and linearly independent columns, we can write $\mathbf{A}$ as the weighted sum of outer products.

Theorem 5.2 (Dyadic Singular Value Decomposition, full-rank $\mathbf{A}$)**.** *Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$ has $\operatorname{rank}(\mathbf{A}) = n$, with $m \geq n$. Then we can write*

$$\mathbf{A} = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T, \tag{5.3}$$

where the columns $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^m$ are orthonormal and $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^n$ are orthonormal, and

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



The expression (5.3) is called the *dyadic form of the SVD*. The matrices $\mathbf{u}_j\mathbf{v}_j^T \in \mathbb{R}^{m\times n}$ are not generally projectors (as were the $\mathbf{v}_j\mathbf{v}_j^T$ matrices in the symmetric spectral decomposition (4.5)), but they have a similar interpretation. When computing the action of $\mathbf{A}$ upon a vector $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{A}\mathbf{x} = \left(\sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T\right)\mathbf{x} = \sum_{j=1}^{n} \sigma_j (\mathbf{v}_j^T\mathbf{x})\mathbf{u}_j,$$

the scalars $\mathbf{v}_j^T\mathbf{x}$ are the coefficients of the *best approximation to* $\mathbf{x}$ *from* $\operatorname{span}\{\mathbf{v}_j\}$. (This value $\mathbf{v}_j^T\mathbf{x}$ tells us "how rich $\mathbf{x}$ is in the $\mathbf{v}_j$ direction.") Now this coefficient $\mathbf{v}_j^T\mathbf{x}$ is multiplied against the *left singular vector* $\mathbf{u}_j$, and scaled by the *singular value* σ_j.

Because we have ordered $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, the leading terms in this sum dominate the others. This fact plays a crucial role in applications where we want to approximate a matrix with its leading low-rank part.

Example 5.2. We continue with Example 5.1. The *dyadic form of the*

SVD takes the form $\mathbf{A} = \sum_{j=1}^{2} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$:

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

5.3 *A first application*

Now that we have the basic SVD in hand, let us consider a very simple way to deploy it: using the SVD to solve a system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

in the case that $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a square, invertible matrix. While the SVD is a fancy tool for solving such a simple equation, this example will give us the first instance of an all-purpose tool to solve a wide variety of regression-type problems.

Assuming $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an invertible square matrix ($m = n$), we have the SVD

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

To solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ first substitute the SVD for $\mathbf{A}$:

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{x} = \mathbf{b}. \tag{5.4}$$

We want to expose the solution $\mathbf{x}$ on the right-hand side. Since the columns of $\mathbf{U}$ are orthonormal, $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, so premultiplying both sides of (5.4) by $\mathbf{U}^T$ gives

$$\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{x} = \mathbf{U}^T\mathbf{b} \quad \Longrightarrow \quad \mathbf{\Sigma}\mathbf{V}^T\mathbf{x} = \mathbf{U}^T\mathbf{b}. \tag{5.5}$$

Since the columns of $\mathbf{A}$ are linearly independent, all the diagonal elements in $\mathbf{\Sigma}$ are nonzero, so

$$\mathbf{\Sigma}^{-1} = \begin{bmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n} \end{bmatrix}.$$

Premultiply each side of (5.5) by $\mathbf{\Sigma}^{-1}$ to get

$$\mathbf{\Sigma}^{-1}\mathbf{\Sigma}\mathbf{V}^T\mathbf{x} = \mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} \quad \Longrightarrow \quad \mathbf{V}^T\mathbf{x} = \mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}. \tag{5.6}$$

Since $\mathbf{V} \in \mathbb{R}^{n \times n}$ is a square matrix and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, we see that $\mathbf{V}^{-1} = \mathbf{V}^T$, and hence $\mathbf{V}\mathbf{V}^T = \mathbf{I}$. Thus premultiplying (5.6) by $\mathbf{V}$ yields

$$\mathbf{V}\mathbf{V}^T\mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} \quad \Longrightarrow \quad \mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}. \tag{5.7}$$

Variants of this equation $\mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$ will recur throughout the next few chapters. Just as $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ has a dyadic form, so too does $\mathbf{A}^{-1} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T$:

$$\mathbf{A} = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T, \qquad \mathbf{A}^{-1} = \sum_{j=1}^{n} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T.$$

We could have arrived at this same equation for $\mathbf{x}$ by computing

$$\begin{aligned}\mathbf{A}^{-1} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^{-1} \\ &= (\mathbf{V}^T)^{-1}\mathbf{\Sigma}^{-1}\mathbf{U}^{-1} \\ &= \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T,\end{aligned}$$

and then writing

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}.$$

We can use this dyadic form of $\mathbf{A}^{-1}$ to write out the solution $\mathbf{x}$:

$$\mathbf{x} \;=\; \mathbf{A}^{-1}\mathbf{b} \;=\; \left(\sum_{j=1}^{n} \frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j^T\right)\mathbf{b} \;=\; \sum_{j=1}^{n} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\,\mathbf{v}_j.$$

This form for the solution will turn out to be very important in this course, so we single it out for attention:

$$\mathbf{x} \;=\; \mathbf{A}^{-1}\mathbf{b} \;=\; \sum_{j=1}^{n} \left(\frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\right)\mathbf{v}_j. \tag{5.8}$$

This vector $\mathbf{x} \in \mathbb{R}^n$ solves $\mathbf{A}\mathbf{x} = \mathbf{b}$ when $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a square invertible matrix.

Can we push this idea a little bit? Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a *rectangular matrix*, still with linearly independent columns. The equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ implies that $\mathbf{b} \in \mathcal{R}(\mathbf{A})$. If this is indeed the case, then the steps we just executed for the square matrix $\mathbf{A}$ still hold: $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ since the columns of $\mathbf{U} \in \mathbb{R}^{m\times n}$ are orthogonal, $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ is remains invertible, and $\mathbf{V} \in \mathbb{R}^{n\times n}$ has $\mathbf{V}^T$ as its inverse, and so the expansion for $\mathbf{x}$ in (5.8) still holds:

Recall that the range (column space) of $\mathbf{A} \in \mathbb{R}^{m\times n}$,

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\},$$

is the set of all linear combinations of the columns of $\mathbf{A}$.

$$\mathbf{x} = \sum_{j=1}^{n} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j.$$

Example 5.3. Pick up the matrix from Example 5.1, with a $\mathbf{b} \in \mathcal{R}(\mathbf{A})$:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ \sqrt{2} & -\sqrt{2} \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 8 \\ 0 \\ 2\sqrt{2} \end{bmatrix}.$$

Recall from Example 5.1 that

$$\mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}, \qquad \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

and

$$\mathbf{u}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \qquad \mathbf{u}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

with $\sigma_1 = 2$ and $\sigma_2 = \sqrt{2}$.

Now use the SVD for $\mathbf{A}$ to write

$$\begin{aligned}\mathbf{x} &= \left(\frac{\mathbf{u}_1^T\mathbf{b}}{\sigma_1}\right)\mathbf{v}_1 + \left(\frac{\mathbf{u}_2^T\mathbf{b}}{\sigma_2}\right)\mathbf{v}_2 \\ &= \frac{2\sqrt{2}}{2}\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} + \frac{8}{\sqrt{2}}\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \;=\; \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}.\end{aligned}$$

Indeed, you can confirm that $\mathbf{A}\mathbf{x} = \mathbf{b}$, as promised.

This is all well and good, provided $\mathbf{b} \in \mathcal{R}(\mathbf{A})$. *But what if* $\mathbf{b} \in \mathcal{R}(\mathbf{A})$*?* Then we cannot satisfy $\mathbf{A}\mathbf{x} = \mathbf{b}$ with any $\mathbf{x}$, and so we must settle for some $\mathbf{x}$ that gives an approximate solution,

$$\mathbf{A}\mathbf{x} \approx \mathbf{b}.$$

How can we select a good approximation $\mathbf{x}$? Naturally we want $\mathbf{Ax}$ to be the vector closest to $\mathbf{b}$, in the sense introduced in Chapter 2:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{Ax}-\mathbf{b}\|.$$

This optimization is called a *least squares problem*. To solve it, we must venture beyond $\mathcal{R}(\mathbf{A})$. We will return to this problem in Section 5.9; we will get some help from an expanded version of the SVD.

5.4 The Full SVD

We still require $\mathbf{A} \in \mathbb{R}^{m\times n}$ to be a rectangular matrix with $m \geq n$ and linearly independent columns. In the reduced SVD, the matrix $\mathbf{U} \in \mathbb{R}^{m\times n}$ had orthonormal columns, so $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, but if $m > n$ the columns of $\mathbf{U}$ *cannot form an orthonormal basis for* $\mathbb{R}^m$, so $\mathbf{U}\mathbf{U}^T \neq \mathbf{I}$.

Here we seek to augment the matrix $\mathbf{U} \in \mathbb{R}^{m\times n}$ with $m-n$ additional column vectors, to give a full set of m orthonormal vectors in $\mathbb{R}^m$. To find these extra vectors: For $j = n+1,\ldots,m$, pick

$$\mathbf{u}_j \perp \text{span}\{\mathbf{u}_1,\ldots,\mathbf{u}_{j-1}\}$$

with $\mathbf{u}_j^T\mathbf{u}_j = 1$. Then define

In Section 5.5, we will see that $\mathbf{u}_{n+1},\ldots,\mathbf{u}_m$ form an orthonormal basis for $\mathcal{N}(\mathbf{A}^T)$, suggesting a convenient way to compute these vectors.

$$\widetilde{\mathbf{U}} = \begin{bmatrix} | & & | & | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_n & \mathbf{u}_{n+1} & \cdots & \mathbf{u}_m \\ | & & | & | & & | \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{5.9}$$

We have constructed $\mathbf{u}_1,\ldots,\mathbf{u}_m$ to be orthonormal vectors, so

$$\mathbf{U}^T\mathbf{U} = \mathbf{I}.$$

However, since $\mathbf{U} \in \mathbb{R}^{m\times m}$, this orthogonality also implies $\mathbf{U}^{-1} = \mathbf{U}^T$.

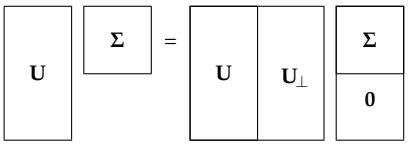
Now we are ready to replace the rectangular matrix $\mathbf{U} \in \mathbb{R}^{m\times n}$ in the reduced SVD (5.2) with the square matrix $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times m}$. To do so, we also need to replace $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ by some $\widetilde{\mathbf{\Sigma}} \in \mathbb{R}^{m\times n}$ in such a way that

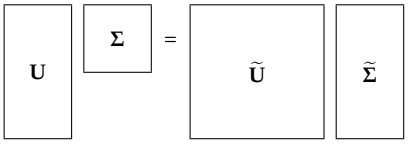
$$\mathbf{U}\mathbf{\Sigma} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}.$$

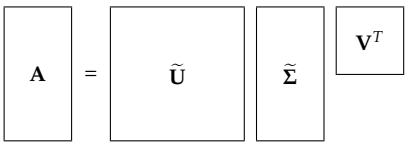
The simplest approach is to obtain $\widetilde{\mathbf{\Sigma}}$ by appending zeros to the end of $\mathbf{\Sigma}$, thus ensuring there is no contribution when the new entries of $\widetilde{\mathbf{U}}$ multiply against the new entries of $\widetilde{\mathbf{\Sigma}}$:

$$\widetilde{\mathbf{\Sigma}} = \begin{bmatrix} \mathbf{\Sigma} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{m\times n}. \tag{5.10}$$

Finally, we arrive at the main result, the *full singular value decomposition*, for the case where $\text{rank}(\mathbf{A}) = n$.







Theorem 5.3 (Full Singular Value Decomposition, full-rank $\mathbf{A}$)**.** *Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$ has* $\text{rank}(\mathbf{A}) = n$*, with $m \geq n$. Then we can write*

$$\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\mathbf{V}^T,$$

where the columns of $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are orthonormal,

$$\widetilde{\mathbf{U}}^T\widetilde{\mathbf{U}} = \mathbf{I} \in \mathbb{R}^{m\times m}, \qquad \mathbf{V}^T\mathbf{V} = \mathbf{I} \in \mathbb{R}^{n\times n},$$

and $\widetilde{\mathbf{\Sigma}} \in \mathbb{R}^{m\times n}$ is zero everywhere except for entries on the main diagonal, where the (j,j) entry is σ_j, for $j = 1,\ldots,n$ and

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

The full SVD is obtained via the Python/NumPy command

```
U, S, Vt = np.linalg.svd(A, full_matrices=True),
```

Definition 5.1. *Let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be a full singular value decomposition. The diagonal entries of $\mathbf{\Sigma}$, denoted $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, are called the* singular values *of $\mathbf{A}$. The columns $\mathbf{u}_1,\ldots,\mathbf{u}_m$ of $\mathbf{U}$ are the* left singular vectors*; the columns $\mathbf{v}_1,\ldots,\mathbf{v}_m$ of $\mathbf{V}$ are the* right singular vectors*.*

5.5 *The Singular Value Decomposition: General $m \geq n$ case*

We have computed the singular value decomposition of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with n linearly independent columns, which gave n positive singular values

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

What happens if the columns of $\mathbf{A}$ are linearly dependent?

If the columns of $\mathbf{A}$ are linearly dependent, then by definition there must exist some matrix $\mathbf{v} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{v} = \mathbf{0}$. Premultiply this equation by $\mathbf{A}^T$ to obtain

$$\mathbf{A}^T\mathbf{A}\mathbf{v} = \mathbf{A}^T\mathbf{0} = \mathbf{0} = 0\mathbf{v},$$

meaning $\mathbf{v}$ *is an eigenvector of $\mathbf{A}^T\mathbf{A}$ corresponding to the eigenvalue $\lambda = 0$.*

Let r denote the number of nonzero eigenvalues of $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n\times n}$. If $r = n$, we are in the full-rank case addressed above. At the other extreme, $r = 0$, we must have the trivial case $\mathbf{A} = \mathbf{0}$. Between these extremes, $0 < r < n$, we label the eigenvalues of $\mathbf{A}^T\mathbf{A}$ as

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0 = \lambda_{r+1} = \cdots = \lambda_n,$$

For $\mathbf{A} = \mathbf{0} \in \mathbb{R}^{m\times n}$, the full SVD is

$$\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^T,$$

where $\widetilde{\mathbf{\Sigma}} = \mathbf{0} \in \mathbb{R}^{m\times n}$ and $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times m}$ and $\widetilde{\mathbf{V}} \in \mathbb{R}^{n\times n}$ are arbitrary matrices with orthonormal columns; natural choices: $\widetilde{\mathbf{U}} = \mathbf{I} \in \mathbb{R}^{m\times m}$, $\widetilde{\mathbf{V}} = \mathbf{I} \in \mathbb{R}^{n\times n}$.

with orthonormal eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

We proceed to build the SVD using the same steps as before, but with a modification in how we handle the left singular vectors.

Note that if $n - r > 1$, be sure to choose *orthonormal* eigenvectors $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$ associated with $\lambda_{r+1} = \cdots = \lambda_n = 0$.

Step 1. Compute the eigenvalues and eigenvectors of $\mathbf{A}^T\mathbf{A}$.

Let r denote the number of nonzero eigenvalues of $\mathbf{A}^T\mathbf{A}$, which thus has $n - r$ zero eigenvalues:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0, \qquad \lambda_{r+1} = \cdots = \lambda_n = 0.$$

The corresponding orthonormal eigenvectors are $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

As we shall see below in Lemma 5.2, the zero eigenvectors of $\mathbf{A}^T\mathbf{A}$ form an orthonormal basis for $\mathcal{N}(\mathbf{A})$, giving a helpful way to compute $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$.

> The vectors $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$ form an orthonormal basis for $\mathcal{N}(\mathbf{A})$.
> To find these vectors, solve $\mathbf{Av} = \mathbf{0}$ and choose your "free variables" to give orthonormal vectors.
> These vectors will be automatically orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_r$.

Step 2. Define $\sigma_j = \|\mathbf{Av}_j\| = \sqrt{\lambda_j}$, $j = 1, \ldots, n$.

This step proceeds without any alterations, though now we have

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0, \qquad \sigma_{r+1} = \cdots = \sigma_n = 0.$$

Step 3a. Define $\mathbf{u}_j = \mathbf{Av}_j/\sigma_j$ for $j = 1, \ldots, r$.

As before, this construction generates orthonormal vectors

$$\mathbf{u}_1, \ldots, \mathbf{u}_r,$$

but note that we only have $j = 1, \ldots, r$ now. (The formula for $\mathbf{u}_j$ fails when $j > r$, since we would divide by $\sigma_j = 0$.)

Step 3b. Construct $\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m$ to be any orthonormal set of vectors that are orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_r$.

This step can be skipped if you are only computing the *reduced* or *dyadic* form of the SVD.

Like the vectors in Step 3a, note that these vectors also satisfy

$$\mathbf{Av}_j = \sigma_j \mathbf{u}_j,$$

since $\mathbf{Av}_j = \mathbf{0} = 0\mathbf{u}_j$ for any choice of $\mathbf{u}_j$, when $j > r$.

This step just extends our construction of the vectors $\mathbf{u}_{n+1}, \ldots, \mathbf{u}_m$ to fill out the $\widetilde{\mathbf{U}}$ matrix for the *full* SVD for full-rank $\mathbf{A}$ in Section 5.4.

The instructions in the blue box do not suggest a convenient way to find $\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m$. Lemma 5.1 below suggests a helpful technique.

> The vectors $\mathbf{u}_{r+1}, \ldots, \mathbf{u}_n$ form an orthonormal basis for $\mathcal{N}(\mathbf{A}^T)$.
> To find these vectors, solve $\mathbf{A}^T\mathbf{u} = \mathbf{0}$ and choose your "free variables" to give orthonormal vectors.
> These vectors will be automatically orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_r$.

Step 4. **Put the pieces together.**

As noted above, the $\sigma_j = 0$ values do not complicate the essential equation

$$\mathbf{A}\mathbf{v}_j = \sigma_j \mathbf{u}_j, \quad j = 1, \ldots, n.$$

Stack these n vector equations as columns of a single matrix equation:

$$\begin{bmatrix} | & & | & | & & | \\ \mathbf{A}\mathbf{v}_1 & \cdots & \mathbf{A}\mathbf{v}_r & \mathbf{A}\mathbf{v}_{r+1} & \cdots & \mathbf{A}\mathbf{v}_n \\ | & & | & | & & | \end{bmatrix} = \begin{bmatrix} | & & | & | & & | \\ \sigma_1\mathbf{u}_1 & \cdots & \sigma_r\mathbf{u}_r & \sigma_{r+1}\mathbf{u}_{r+1} & \cdots & \sigma_n\mathbf{u}_n \\ | & & | & | & & | \end{bmatrix}$$

$$= \begin{bmatrix} | & & | & | & & | \\ \sigma_1\mathbf{u}_1 & \cdots & \sigma_r\mathbf{u}_r & \mathbf{0} & \cdots & \mathbf{0} \\ | & & | & | & & | \end{bmatrix},$$

where this last line follows from $\sigma_{r+1} = \cdots = \sigma_n = 0$. Just as we did in the $r = n$ case above, we can tease each side of this equation apart:

$$\mathbf{A}\begin{bmatrix} | & & | & | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r & \mathbf{v}_{r+1} & \cdots & \mathbf{v}_n \\ | & & | & | & & | \end{bmatrix} = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ | & & | \end{bmatrix} \left[\begin{array}{ccc|ccc} \sigma_1 & & & 0 & \cdots & 0 \\ & \ddots & & \vdots & \ddots & \vdots \\ & & \sigma_r & 0 & \cdots & 0 \end{array}\right] \begin{matrix} \\ \\ \\ \end{matrix} \Bigg\} r$$

$$\underbrace{\qquad\qquad}_{r} \quad \underbrace{\qquad\qquad}_{n-r}$$

Divide the *right singular vectors* into two matrices:

$$\mathbf{V} := \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r \\ | & & | \end{bmatrix} \in \mathbb{R}^{n\times r}, \qquad \mathbf{V}_\perp := \begin{bmatrix} | & & | \\ \mathbf{v}_{r+1} & \cdots & \mathbf{v}_n \\ | & & | \end{bmatrix} \in \mathbb{R}^{n\times(n-r)},$$

which we concatenate into the $n \times n$ matrix $\widetilde{\mathbf{V}}$ with orthonormal columns:

$$\widetilde{\mathbf{V}} := [\mathbf{V} \;\; \mathbf{V}_\perp] \in \mathbb{R}^{n\times n}, \qquad \widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}} = \mathbf{I}, \quad \widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^T = \mathbf{I}.$$

Stack the first r *left singular vectors* and *singular values* into the matrices

$$\mathbf{U} := \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ | & & | \end{bmatrix} \in \mathbb{R}^{m\times r}, \qquad \mathbf{\Sigma} := \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}.$$

With these definitions in hand, the long equation above reduces to

$$\mathbf{A}\widetilde{\mathbf{V}} = \mathbf{U}\,[\mathbf{\Sigma}\ \ \mathbf{0}].$$

Multiply both sides by $\widetilde{\mathbf{V}}^T$ on the right to get

$$\mathbf{A} = \mathbf{U}\,[\mathbf{\Sigma}\ \ \mathbf{0}]\widetilde{\mathbf{V}}^T.$$

$\mathbf{A}\widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^T = \mathbf{A}\mathbf{I} = \mathbf{A}.$

Since we partition $\widetilde{\mathbf{V}} = [\mathbf{V}\ \ \mathbf{V}_\perp]$, we have

$$\mathbf{A} = \mathbf{U}\begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{V} & \mathbf{V}_\perp \end{bmatrix}^T = \mathbf{U}\begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{V}^T \\ \mathbf{V}_\perp^T \end{bmatrix} = \mathbf{U}\left(\mathbf{\Sigma}\mathbf{V}^T + \mathbf{0}\mathbf{V}_\perp^T\right) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

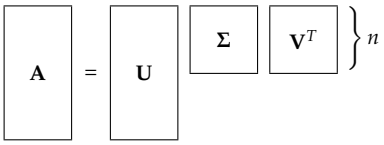
Thus when the dust settles, we are left with the beautiful factorization

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

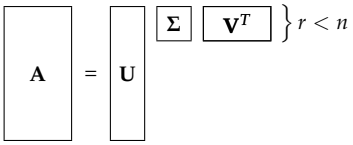
which looks just like the *reduced SVD* we obtained in the full-rank case, but with slightly different definitions and dimensions:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ | & & | \end{bmatrix}\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}\begin{bmatrix} — & \mathbf{v}_1^T & — \\ & \vdots & \\ — & \mathbf{v}_r^T & — \end{bmatrix}.$$

Reduced SVD, full-rank case ($r = n$)



Reduced SVD, rank-deficient case ($r < n$)



We emphasize the dimensions here, as illustrated on the right:

$$\mathbf{U} \in \mathbb{R}^{m\times r}, \qquad \mathbf{\Sigma} \in \mathbb{R}^{r\times r}, \qquad \mathbf{V} \in \mathbb{R}^{n\times r}.$$

The product $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is still an $m \times n$ matrix, despite the dimension r that arises in the interior of the product.

The reduced SVD in the last blue box immediately suggests a *dyadic form* for the case of rank-deficient $\mathbf{A}$. It looks just like equation (5.3), except for the top limit on the sum is r instead of n:

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{5.11}$$

What about the full SVD? The construction is a little clunkier, but not in an especially difficult (or interesting) way. For the right singular vectors we already have

$$\mathbf{V} := \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r \\ | & & | \end{bmatrix} \in \mathbb{R}^{n\times r}, \qquad \mathbf{V}_\perp := \begin{bmatrix} | & & | \\ \mathbf{v}_{r+1} & \cdots & \mathbf{v}_n \\ | & & | \end{bmatrix} \in \mathbb{R}^{n\times(n-r)}, \qquad \widetilde{\mathbf{V}} := \begin{bmatrix} \mathbf{V} & \mathbf{V}_\perp \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

Now define the analogous matrices for the left singular vectors:

$$\mathbf{U} := \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ | & & | \end{bmatrix} \in \mathbb{R}^{m\times r}, \qquad \mathbf{U}_\perp := \begin{bmatrix} | & & | \\ \mathbf{u}_{r+1} & \cdots & \mathbf{u}_m \\ | & & | \end{bmatrix} \in \mathbb{R}^{m\times(m-r)}, \qquad \widetilde{\mathbf{U}} := \begin{bmatrix} \mathbf{U} & \mathbf{U}_\perp \end{bmatrix} \in \mathbb{R}^{m\times m}.$$

Finally, create an $m \times n$ matrix containing the singular values, padded with zeros to make up the dimensions:

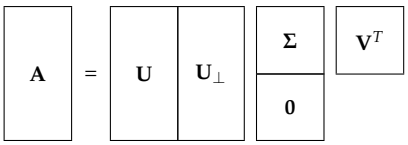
$$\mathbf{\Sigma} := \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}, \qquad \widetilde{\mathbf{\Sigma}} := \underbrace{\begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\begin{matrix} r & n-r \end{matrix}} \begin{matrix} \} r \\ \} m-r \end{matrix}$$

With the right definitions, the full SVD takes a simple form:

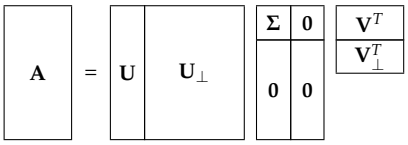
$$\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^T. \tag{5.12}$$

Please pay special attention to those two sketches on the right. Understanding these diagrams goes a long way toward understanding the full singular value decomposition.

Full SVD, full-rank case ($r = n \leq m$)



Full SVD, rank-deficient case ($r < n \leq m$)



Example 5.4. Consider the matrix 3×2 matrix ($m = 3$, $n = 2$)

$$\mathbf{A} = \begin{bmatrix} 1 & \sqrt{2} \\ 1 & \sqrt{2} \\ 1 & \sqrt{2} \end{bmatrix}.$$

Note that the second column is a multiple of the first, so the columns are *linear dependent*.

- Step 1. Form

  $$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 3 & 3\sqrt{2} \\ 3\sqrt{2} & 6 \end{bmatrix}$$

  having characteristic polynomial

  $$\det(\lambda\mathbf{I} - \mathbf{A}^T\mathbf{A}) = \lambda^2 - 9\lambda = \lambda(\lambda - 9).$$

  Label the eigenvalues

  $$\lambda_1 = 9, \qquad \lambda_2 = 0,$$

  with corresponding orthonormal eigenvectors

  $$\mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{3} \\ \sqrt{2}/\sqrt{3} \end{bmatrix}, \qquad \mathbf{v}_2 = \begin{bmatrix} \sqrt{2}/\sqrt{3} \\ -1/\sqrt{3} \end{bmatrix}.$$

  These vectors are the *right singular vectors*. Since $\mathbf{A}^T\mathbf{A}$ has only one nonzero eigenvalue, we conclude that $r = 1$.

- Step 2. Define the singular values

  $$\sigma_1 = \sqrt{\lambda_1} = 3, \qquad \sigma_2 = \sqrt{\lambda_2} = 0.$$

- Step 3a. Since $r = 1$, we can define the *left singular vector* via

$$\mathbf{u}_1 = \frac{1}{\sigma_1}\mathbf{A}\mathbf{v}_1 = \frac{1}{3}\begin{bmatrix} 3/\sqrt{3} \\ 3/\sqrt{3} \\ 3/\sqrt{3} \end{bmatrix} = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix},$$

  which is a unit vector, $\|\mathbf{u}_1\| = 1$.

- Step 3b. To compute the remaining $m - r = 3 - 1 = 2$ *left singular vectors*, we must find two mutually orthogonal unit vectors that are also orthogonal to $\mathbf{u}_1$. Following the hint in the black box above, we look for vectors for which $\mathbf{A}^T\mathbf{u} = \mathbf{0}$, i.e.,

$$\begin{bmatrix} 1 & 1 & 1 \\ \sqrt{2} & \sqrt{2} & \sqrt{2} \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

  This equation reduces to the single equation

$$\alpha + \beta + \gamma = 0.$$

  Let α and β be free variables, and solve for

$$\gamma = -\alpha - \beta.$$

  Thus the vectors $\mathbf{u}_2$ and $\mathbf{u}_3$ must be orthonormal vectors of the form

$$\mathbf{u} = \begin{bmatrix} \alpha \\ \beta \\ -\alpha - \beta \end{bmatrix}.$$

  For example, pick $\alpha = 1/\sqrt{2}$ and $\beta = 0$ to get the unit vector

$$\mathbf{u}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix}.$$

  You can make other choices of α and β, which would lead to a different $\mathbf{u}_2$. For example, you could choose $\alpha = 0$ and $\beta = 1/\sqrt{2}$, giving a different $\mathbf{u}_2$ and, eventually, a different $\mathbf{u}_3$ too. So long as $\mathbf{u}_2$ and $\mathbf{u}_3$ are orthonormal unit vectors that are also orthogonal to $\mathbf{u}_1$, you have a valid set of vectors.

  To find $\mathbf{u}_3$, eliminate one of the free variables by adding the additional requirement that $0 = \mathbf{u}_2^T\mathbf{u}$, i.e.,

$$0 = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \\ -\alpha - \beta \end{bmatrix} = \frac{\alpha + \alpha + \beta}{\sqrt{2}},$$

  and hence $2\alpha + \beta = 0$, i.e., $\beta = -2\alpha$:

$$\mathbf{u} = \begin{bmatrix} \alpha \\ -2\alpha \\ \alpha \end{bmatrix}.$$

Pick $\alpha = 1/\sqrt{6}$ to get

$$\mathbf{u}_3 = \begin{bmatrix} 1/\sqrt{6} \\ -2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix},$$

which indeed is a unit vector that is orthogonal to $\mathbf{u}_1$ and $\mathbf{u}_2$.

Now we have all the ingredients on hand to form the *reduced SVD* from

$$\mathbf{U} = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}, \qquad \mathbf{\Sigma} = \begin{bmatrix} 3 \end{bmatrix}, \qquad \mathbf{V} = \begin{bmatrix} 1/\sqrt{3} \\ \sqrt{2}/\sqrt{3} \end{bmatrix},$$

giving

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix} \begin{bmatrix} 3 \end{bmatrix} \begin{bmatrix} 1/\sqrt{3} & \sqrt{2}/\sqrt{3} \end{bmatrix}.$$

The *dyadic SVD* has just $r = 1$ term in the sum:

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T = 3 \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix} \begin{bmatrix} 1/\sqrt{3} & \sqrt{2}/\sqrt{3} \end{bmatrix}.$$

The *full SVD* requires the augmented matrices

$$\widetilde{\mathbf{U}} = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{6} \\ 1/\sqrt{3} & 0 & -2/\sqrt{6} \\ 1/\sqrt{3} & -1/\sqrt{2} & 1/\sqrt{6} \end{bmatrix}, \qquad \widetilde{\mathbf{\Sigma}} = \begin{bmatrix} 3 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \widetilde{\mathbf{V}} = \begin{bmatrix} 1/\sqrt{3} & \sqrt{2}/\sqrt{3} \\ \sqrt{2}/\sqrt{3} & -1/\sqrt{3} \end{bmatrix},$$

giving

$$\begin{bmatrix} 1 & \sqrt{2} \\ 1 & \sqrt{2} \\ 1 & \sqrt{2} \end{bmatrix} = \mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^T = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{6} \\ 1/\sqrt{3} & 0 & -2/\sqrt{6} \\ 1/\sqrt{3} & -1/\sqrt{2} & 1/\sqrt{6} \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{3} & \sqrt{2}/\sqrt{3} \\ \sqrt{2}/\sqrt{3} & -1/\sqrt{3} \end{bmatrix}.$$

5.6 *Modification for the case of $m < n$*

How does the singular value decomposition change if $\mathbf{A}$ has more columns than rows, $n > m$? The answer is easy: write the SVD of $\mathbf{A}^T$ (which has more rows than columns) using the procedure above, then take the transpose of each term in the SVD. If this makes good sense, skip ahead to the next section. If you prefer the gory details, read on.

We now formally adapt the steps described above to handle the case $n > m$. First off, note that $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n \times n}$, while $\mathbf{A}\mathbf{A}^T \in \mathbb{R}^{m \times m}$: we

prefer to work with the smaller matrix (in this case $\mathbf{A}\mathbf{A}^T$), since the eigenvalue problem will be easier to solve.

Step 1. **Compute the eigenvalues and eigenvectors of $\mathbf{A}\mathbf{A}^T$.**

Label the eigenvalues of $\mathbf{A}\mathbf{A}^T \in \mathbb{R}^{m\times m}$ as

$$\lambda_1 \geq \cdots \geq \lambda_r > 0 = \lambda_{r+1} = \cdots = \lambda_m,$$

and corresponding orthonormal eigenvectors

$$\mathbf{u}_1, \ldots, \mathbf{u}_r, \mathbf{u}_{r+1}, \ldots, \mathbf{u}_m.$$

We have defined r to be the number of nonzero eigenvalues of $\mathbf{A}\mathbf{A}^T$.

Step 2. **Define $\sigma_j = \|\mathbf{A}^T\mathbf{u}_j\| = \sqrt{\lambda_j}$, $j = 1, \ldots, m$.**

Step 3a. **Define $\mathbf{v}_j = \mathbf{A}^T\mathbf{u}_j/\sigma_j$ for $j = 1, \ldots, r$.**

Step 3b. **Construct $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$ to be any orthonormal set of vectors that are orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_r$.**

The vectors $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$ form an orthonormal basis for $\mathcal{N}(\mathbf{A})$.
To find these vectors, solve $\mathbf{A}\mathbf{v} = \mathbf{0}$ and choose your
"free variables" to give orthonormal vectors.
These vectors will be automatically orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_r$.

Step 4. **Put the pieces together.**

First, defining

$$\mathbf{U} = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ | & & | \end{bmatrix} \in \mathbb{R}^{m\times r}, \qquad \mathbf{V} = \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r \\ | & & | \end{bmatrix} \in \mathbb{R}^{n\times r},$$

with diagonal matrix

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r},$$

we have the reduced SVD

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

To obtain the full SVD, extend $\mathbf{U} \in \mathbb{R}^{m\times r}$ to $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times m}$ by appending the vectors $\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m$:

$$\widetilde{\mathbf{U}} = \begin{bmatrix} \mathbf{U} & \mathbf{U}_\perp \end{bmatrix} = \begin{bmatrix} | & & | & | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r & \mathbf{u}_{r+1} & \cdots & \mathbf{u}_m \\ | & & | & | & & | \end{bmatrix} \in \mathbb{R}^{m\times m},$$

extend $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ to

$$\widetilde{\mathbf{\Sigma}} := \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{matrix} \} r \\ \} m-r \end{matrix}$$
$$\underbrace{\phantom{\mathbf{\Sigma}}}_{r} \quad \underbrace{\phantom{\mathbf{0}}}_{n-r}$$

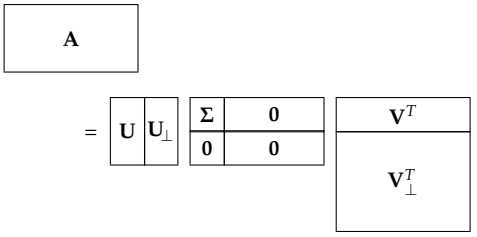
and extend $\widetilde{\mathbf{V}} \in \mathbb{R}^{n\times n}$ by appending the vectors $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$:

$$\widetilde{\mathbf{V}} = \begin{bmatrix} \mathbf{V} & \mathbf{V}_\perp \end{bmatrix} = \begin{bmatrix} | & & | & | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r & \mathbf{v}_{r+1} & \cdots & \mathbf{v}_n \\ | & & | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

We thus arrive at the full SVD,

$$\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^T.$$

Full SVD, rank-deficient case ($r < m \leq n$)



Example 5.5. Consider the matrix

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

Since $2 = m < n = 3$, we choose to work with the smaller matrix

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 6 & -6 \\ -6 & 6 \end{bmatrix},$$

which has eigenvalues $\lambda_1 = 12$ and $\lambda_2 = 0$. (Clearly the second row of $\mathbf{A}$ is a multiple of the first row, so we expect that zero eigenvalue.) The singular vectors are thus $\sigma_1 = \sqrt{12}$ and $\sigma_2 = 0$. Compute the orthonormal eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$, then proceed to create $\mathbf{v}_1 = \mathbf{A}^T\mathbf{u}_1/\sigma_1$ and then generate two more orthonormal vectors $\mathbf{v}_2$ and $\mathbf{v}_3$ that are orthogonal to $\mathbf{v}_1$. (The vectors $\mathbf{v}_2$ and $\mathbf{v}_3$ must both be members of $\mathcal{N}(\mathbf{A})$, giving an easy way to generate vectors orthogonal to $\mathbf{v}_1$.) You should arrive at the reduced SVD

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \sqrt{12} \end{bmatrix} \begin{bmatrix} 1/\sqrt{6} & 2/\sqrt{6} & 1/\sqrt{6} \end{bmatrix}$$

the dyadic form

$$\mathbf{A} = \sigma_1\mathbf{u}_1\mathbf{v}_1^T = \sqrt{12} \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{6} & 2/\sqrt{6} & 1/\sqrt{6} \end{bmatrix},$$

If instead you decided to work with the 3×3 matrix

$$\begin{bmatrix} 2 & 4 & 2 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix},$$

you have a more elaborate eigenvalue problem to solve. Ultimately you would find $\lambda_1 = 12$, $\lambda_2 = \lambda_3 = 0$, consistent with the eigenvalues found from $\mathbf{A}\mathbf{A}^T$.

and the full SVD

$$\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{6} & 2/\sqrt{6} & 1/\sqrt{6} \\ -5/\sqrt{30} & 2/\sqrt{30} & 1/\sqrt{30} \\ 0 & 1/\sqrt{5} & -2/\sqrt{5} \end{bmatrix}.$$

5.7 *General statement of the singular value decomposition*

We now state the singular value decomposition in full generality.

Theorem 5.4 (Singular Value Decomposition). *Suppose* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *has* $\operatorname{rank}(\mathbf{A}) = r$. *Then there exists matrices* $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times m}$ *and* $\widetilde{\mathbf{V}} \in \mathbb{R}^{n\times n}$, *each having orthonormal columns, which can be partitioned as*

$$\widetilde{\mathbf{U}} = \begin{bmatrix} \mathbf{U} & \mathbf{U}_\perp \end{bmatrix}, \qquad \widetilde{\mathbf{V}} = \begin{bmatrix} \mathbf{V} & \mathbf{V}_\perp \end{bmatrix}$$

with

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_r \end{bmatrix} \in \mathbb{R}^{m\times r}, \quad \mathbf{U}_\perp = \begin{bmatrix} \mathbf{u}_{r+1} & \cdots & \mathbf{u}_n \end{bmatrix} \in \mathbb{R}^{m\times(m-r)}$$

and

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_r \end{bmatrix} \in \mathbb{R}^{n\times r}, \quad \mathbf{V}_\perp = \begin{bmatrix} \mathbf{v}_{r+1} & \cdots & \mathbf{v}_n \end{bmatrix} \in \mathbb{R}^{n\times(n-r)},$$

and a diagonal matrix

$$\widetilde{\mathbf{\Sigma}} := \begin{bmatrix} \underbrace{\mathbf{\Sigma}}_{r} & \underbrace{\mathbf{0}}_{n-r} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{matrix} \}\, r \\ \}\, m-r \end{matrix}$$

with $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$ *for*

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

such that the following expressions for $\mathbf{A}$ *all hold:*

Reduced SVD: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$;

Dyadic SVD: $\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$;

Full SVD: $\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^T$.

Of course, when $r = 0$ all the singular values are zero; when $r = \min\{m, n\}$, all the singular values are positive.

5.8 *Connection to the four fundamental subspaces*

Having labored to develop the singular value decomposition in its complete generality, we are ready to reap its many rewards. We begin

by establishing the connection between the singular vectors and the 'four fundamental subspaces,' i.e., the column space

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m,$$

the row space

$$\mathcal{R}(\mathbf{A}^T) = \{\mathbf{A}^T\mathbf{y} : \mathbf{y} \in \mathbb{R}^m\} \subseteq \mathbb{R}^n,$$

the null space

$$\mathcal{N}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{0}\} \subseteq \mathbb{R}^n,$$

and the left null space

$$\mathcal{N}(\mathbf{A}^T) = \{\mathbf{y} \in \mathbb{R}^m : \mathbf{A}^T\mathbf{y} = \mathbf{0}\} \subseteq \mathbb{R}^m.$$

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, take the dyadic form of the SVD,

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

We can characterize each of the fundamental subspaces in terms of the singular values and singular vectors.

$\mathcal{R}(\mathbf{A}) = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$

To see this, relationship, we will prove that each of the subspaces $\mathcal{R}(\mathbf{A})$ and $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ contain the other.

Let $\mathcal{U}$ and $\mathcal{V}$ be two subspaces. If $\mathcal{U}$ is contained in $\mathcal{V}$ (written $\mathcal{U} \subseteq \mathcal{V}$) and $\mathcal{V}$ is contained in $\mathcal{U}$ (written $\mathcal{V} \subseteq \mathcal{U}$), then $\mathcal{U} = \mathcal{V}$. This technique is the standard way mathematicians show that two sets are the same.

First, we will show that $\mathcal{R}(\mathbf{A}) \subseteq \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ by showing that any vector $\mathbf{A}\mathbf{x} \in \mathcal{R}(\mathbf{A})$ must be in $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$. Apply $\mathbf{A}$ to a generic vector $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{A}\mathbf{x} = \Big(\sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T\Big)\mathbf{x} = \sum_{j=1}^{r} \left(\sigma_j \mathbf{u}_j \mathbf{v}_j^T \mathbf{x}\right) = \sum_{j=1}^{r} \left(\sigma_j \mathbf{v}_j^T \mathbf{x}\right)\mathbf{u}_j, \tag{5.13}$$

where in the last step we have switched the order of the scalar $\mathbf{v}_j^T\mathbf{x}$ and the vector $\mathbf{u}_j$. Equation 5.13 shows that $\mathbf{A}\mathbf{x}$ is a weighted sum of the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_r$, and hence in the span of these vectors. Since this is true for all $\mathbf{A}\mathbf{x}$, we conclude that

$$\mathcal{R}(\mathbf{A}) \subseteq \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}.$$

Can we conclude the converse? We know that $\mathcal{R}(\mathbf{A})$ is a subspace, so if we can show that each of the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_r$ is in $\mathcal{R}(\mathbf{A})$, then we will know that

$$\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\} \subseteq \mathcal{R}(\mathbf{A}). \tag{5.14}$$

To show that $\mathbf{u}_k \in \mathcal{R}(\mathbf{A})$, we must find some $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{u}_k$. Inspect equation (5.13). We can make $\mathbf{A}\mathbf{x} = \mathbf{u}_k$ if all the coefficients $\sigma_j \mathbf{v}_j^T\mathbf{x}$ are zero when $j \neq k$, and $\sigma_k \mathbf{v}_k^T\mathbf{x} = 1$. Can you see how to use

orthogonality of the right singular vectors $\mathbf{v}_1, \ldots, \mathbf{v}_r$ to achieve this? Setting

$$\mathbf{x} = \frac{1}{\sigma_k}\mathbf{v}_k,$$

we have $\mathbf{A}\mathbf{x} = \mathbf{u}_k$. Thus $\mathbf{u}_k \in \mathcal{R}(\mathbf{A})$, and we can conclude that (5.14) holds. Since the subspace $\mathcal{R}(\mathbf{A})$ and $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ contain one another, we conclude that

$$\mathcal{R}(\mathbf{A}) = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}.$$

$$\mathcal{N}(\mathbf{A}^T) = \text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}$$

Together all the left singular vectors $\mathbf{u}_1, \ldots, \mathbf{u}_m$ form an orthonormal set in the m-dimensional space $\mathbb{R}^m$, so

$$\mathbb{R}^m = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}.$$

Since $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ is orthogonal to $\text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}$, and $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\} = \mathcal{R}(\mathbf{A})$,

$\text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}$ is the set of all vectors orthogonal to $\mathcal{R}(\mathbf{A})$.

Lemma 5.1. *For any matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathcal{N}(\mathbf{A}^T) \perp \mathcal{R}(\mathbf{A})$.

The notation $\mathcal{N}(\mathbf{A}^T) \perp \mathcal{R}(\mathbf{A})$ means: "Every vector $\mathbf{y} \in \mathcal{N}(\mathbf{A}^T)$ is orthogonal to every vector $\mathbf{b} \in \mathcal{R}(\mathbf{A})$."

Proof. Suppose that $\mathbf{y} \in \mathcal{N}(\mathbf{A}^T)$, so that $\mathbf{A}^T\mathbf{y} = \mathbf{0}$, and suppose $\mathbf{b} \in \mathcal{R}(\mathbf{A})$, so that there exists some $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}$. Then

$$\mathbf{b}^T\mathbf{y} = (\mathbf{A}\mathbf{x})^T\mathbf{y} = \mathbf{x}^T\mathbf{A}^T\mathbf{y} = \mathbf{x}^T(\mathbf{A}^T\mathbf{y}) = \mathbf{x}^T\mathbf{0} = 0.$$

We thus conclude that every vector in $\mathcal{N}(\mathbf{A}^T)$ is orthgonal to every vector in $\mathcal{R}(\mathbf{A})$, and so $\mathcal{N}(\mathbf{A}^T) \perp \mathcal{R}(\mathbf{A})$. ∎

Lemma 5.1 implies that $\mathcal{N}(\mathbf{A}^T) \subseteq \text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}$, since the latter is the set of everything orthogonal to $\mathcal{R}(\mathbf{A})$. To prove that $\mathcal{N}(\mathbf{A}^T) = \text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}$, we must show that $\mathbf{u}_k \in \mathcal{N}(\mathbf{A}^T)$ for $k = r+1, \ldots, m$. This is easy if we write transpose the dyadic form of the SVD:

$$\mathbf{A}^T = \left(\sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T\right)^T = \sum_{j=1}^{r} \sigma_j (\mathbf{u}_j \mathbf{v}_j^T)^T = \sum_{j=1}^{r} \sigma_j \mathbf{v}_j \mathbf{u}_j^T. \tag{5.15}$$

Thus for $k = r+1, \ldots, m$,

$$\mathbf{A}^T\mathbf{u}_k = \left(\sum_{j=1}^{r} \sigma_j \mathbf{v}_j \mathbf{u}_j^T\right)\mathbf{u}_k = \sum_{j=1}^{r} \sigma_j \mathbf{v}_j (\mathbf{u}_j^T\mathbf{u}_k) = \sum_{j=1}^{r} \sigma_j \mathbf{v}_j \cdot 0 = 0,$$

since $\mathbf{u}_j^T\mathbf{u}_k = 0$ when $1 \le j \le r$ and $r+1 \le k \le m$.

We conclude that $\mathbf{u}_k \in \mathcal{N}(\mathbf{A}^T)$ for all $k = r+1, \ldots, m$ and hence $\text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\} \subseteq \mathcal{N}(\mathbf{A}^T)$. Since we already proved the reverse containment, we have

$$\mathcal{N}(\mathbf{A}^T) = \text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}.$$

Just as the left singular vectors spanned $\mathcal{R}(\mathbf{A})$ and $\mathcal{N}(\mathbf{A}^T)$, so the right singular vectors span $\mathcal{R}(\mathbf{A}^T)$ and $\mathcal{N}(\mathbf{A})$.

$$\mathcal{R}(\mathbf{A}^T) = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$$

To prove this statement, use the same argument used for $\mathcal{R}(\mathbf{A})$ above, but now with the dyadic form

$$\mathbf{A}^T = \sum_{j=1}^{r} \sigma_j \mathbf{v}_j \mathbf{u}_j^T.$$

$$\mathcal{N}(\mathbf{A}) = \text{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}$$

This argument follows exactly as for $\mathcal{N}(\mathbf{A}^T)$, but with $\mathbf{A}$ replacing $\mathbf{A}^T$.

While considering $\mathcal{N}(\mathbf{A})$, we note a helpful connection between $\mathcal{N}(\mathbf{A})$ and the eigenvectors of $\mathbf{A}^T\mathbf{A}$ associated with zero eigenvalues.

Lemma 5.2. *For any matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathcal{N}(\mathbf{A}^T\mathbf{A}) = \mathcal{N}(\mathbf{A})$.

Proof. First we show that $\mathcal{N}(\mathbf{A})$ is contained in $\mathcal{N}(\mathbf{A}^T\mathbf{A})$. If $\mathbf{x} \in \mathcal{N}(\mathbf{A})$, then $\mathbf{A}\mathbf{x} = \mathbf{0}$. Premultiplying by $\mathbf{A}^T$ gives $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{0}$, so $\mathbf{x} \in \mathcal{N}(\mathbf{A}^T\mathbf{A})$.

Now we show that $\mathcal{N}(\mathbf{A}^T\mathbf{A})$ is contained in $\mathcal{N}(\mathbf{A})$. If $\mathbf{x} \in \mathcal{N}(\mathbf{A}^T\mathbf{A})$, then $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{0}$. Premultiplying by $\mathbf{x}^T$ gives

$$\mathbf{0} = \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} = (\mathbf{A}\mathbf{x})^T(\mathbf{A}\mathbf{x}) = \|\mathbf{A}\mathbf{x}\|^2.$$

Since $\|\mathbf{A}\mathbf{x}\| = \mathbf{0}$, we conclude that $\mathbf{A}\mathbf{x} = \mathbf{0}$, and so $\mathbf{x} \in \mathcal{N}(\mathbf{A})$.

Since the spaces $\mathcal{N}(\mathbf{A})$ and $\mathcal{N}(\mathbf{A}^T\mathbf{A})$ each contain the other, we conclude that $\mathcal{N}(\mathbf{A}) = \mathcal{N}(\mathbf{A}^T\mathbf{A})$. ∎

Putting together the key results from this section, we arrive at a comprehensive version of the Fundamental Theorem of Linear Algebra[1] in terms of the SVD.

[1] Gilbert Strang. The Fundamental Theorem of Linear Algebra. *Amer. Math. Monthly*, 100:848–855, 1993

Theorem 5.5 (Fundamental Theorem of Linear Algebra: FTLA)**.** *Suppose* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *has* $\operatorname{rank}(\mathbf{A}) = r$*, with left singular vectors* $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ *and right singular vectors* $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$*. Then*

$$\begin{aligned} \mathcal{R}(\mathbf{A}) &= \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\} \\ \mathcal{N}(\mathbf{A}^T) &= \operatorname{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\} \\ \mathcal{R}(\mathbf{A}^T) &= \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_r\} \\ \mathcal{N}(\mathbf{A}) &= \operatorname{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}, \end{aligned}$$

which implies

$$\begin{aligned} \mathcal{R}(\mathbf{A}) \oplus \mathcal{N}(\mathbf{A}^T) &= \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_m\} = \mathbb{R}^m \\ \mathcal{R}(\mathbf{A}^T) \oplus \mathcal{N}(\mathbf{A}) &= \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \;\; = \mathbb{R}^n, \end{aligned}$$

and

$$\mathcal{R}(\mathbf{A}) \perp \mathcal{N}(\mathbf{A}^T), \qquad \mathcal{R}(\mathbf{A}^T) \perp \mathcal{N}(\mathbf{A}).$$

The notation $\oplus$ is called the *direct sum* of subspaces. Suppose $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$ are subspaces of $\mathbb{R}^n$. To say $\mathcal{U} \oplus \mathcal{V} = \mathcal{W}$ means that "any vector $\mathbf{w} \in \mathcal{W}$ can be written as $\mathbf{w} = \mathbf{u} + \mathbf{v}$ for *unique* vectors $\mathbf{u} \in \mathcal{U}$ and $\mathbf{v} \in \mathcal{V}$. Writing $\mathcal{U} + \mathcal{V} = \mathcal{W}$ means the same thing, but without the *uniqueness* of $\mathbf{u}$ and $\mathbf{v}$.

5.9 *Revisiting linear systems*

In Section 5.3 we used the SVD to tackle the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, which has the solution

$$\mathbf{x} = \sum_{j=1}^{n} \frac{\mathbf{u}_j^T \mathbf{b}}{\sigma_j} \mathbf{v}_j$$

provided $\mathbf{b} \in \mathcal{R}(\mathbf{A})$.

When $\mathbf{b} \notin \mathcal{R}(\mathbf{A})$, we can only hope to solve this equation in the sense of *least squares sense*:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|. \tag{5.16}$$

This problem can be viewed as standard linear least squares regression. In regression, one posits that the data $\mathbf{y} \in \mathbb{R}^n$ satisfy the model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where the *design matrix* $\mathbf{X} \in \mathbb{R}^{n\times p}$ contains the independent variables, $\boldsymbol{\beta} \in \mathbb{R}^p$ contains the (unknown) model parameters, and $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ describes the noise (upon which we might impose statistical assumptions: zero mean, normality, etc.). Standard least squares seeks the model parameters that satisfy

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|.$$

The SVD will give an elegant way to solve this problem as well.

The Fundamental Theorem of Linear Algebra (Theorem 5.5) ensures that $\mathbb{R}^m = \mathcal{R}(\mathbf{A}) \oplus \mathcal{N}(\mathbf{A}^T)$, which means that the vector $\mathbf{b} \in \mathbb{R}^m$ can be written *uniquely* as

$$\mathbf{b} = \mathbf{b}_R + \mathbf{b}_N, \qquad \mathbf{b}_R \in \mathcal{R}(\mathbf{A}), \quad \mathbf{b}_N \in \mathcal{N}(\mathbf{A}^T).$$

Now for any choice of $\mathbf{x} \in \mathbb{R}^n$,

$$\begin{aligned} \mathbf{b} - \mathbf{A}\mathbf{x} &= \mathbf{b}_R + \mathbf{b}_N - \mathbf{A}\mathbf{x} \\ &= (\mathbf{b}_R - \mathbf{A}\mathbf{x}) + \mathbf{b}_N, \end{aligned}$$

where we have grouped terms strategically: By its form we must have $\mathbf{A}\mathbf{x} \in \mathcal{R}(\mathbf{A})$, and since $\mathcal{R}(\mathbf{A})$ is a subspace, the sum of two $\mathcal{R}(\mathbf{A})$

vectors is also in $\mathcal{R}(\mathbf{A})$. The FTLA tells us that $\mathcal{R}(\mathbf{A}) \perp \mathcal{N}(\mathbf{A}^T)$, so we can apply the Pythagorean Theorem to obtain

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 = \|\mathbf{b}_R - \mathbf{A}\mathbf{x}\|^2 + \|\mathbf{b}_N\|^2 \tag{5.17}$$

for any $\mathbf{x}$. To solve the least squares problem (5.16), we must pick $\mathbf{x}$ to minimize the expression (5.17). No choice of $\mathbf{x}$ can reach the $\|\mathbf{b}_N\|^2$ term; on the other hand, we can pick $\mathbf{x}$ to reach any vector $\mathbf{A}\mathbf{x}$ in $\mathcal{R}(\mathbf{A})$. In particular, we can pick $\mathbf{x}$ so that $\mathbf{A}\mathbf{x} = \mathbf{b}_R$, and with this choice

We seek $\mathbf{x}$ that solves $\mathbf{A}\mathbf{x} = \mathbf{b}_R$. As emphasized in Figure 7.1, this equation always has a solution since $\mathbf{b}_R \in \mathcal{R}(\mathbf{A})$.

$$\begin{aligned}\|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 &= \|\mathbf{b}_R - \mathbf{A}\mathbf{x}\|^2 + \|\mathbf{b}_N\|^2 \\ &= \|\mathbf{b}_R - \mathbf{b}_R|^2 + \|\mathbf{b}_N\|^2 \\ &= \|\mathbf{b}_N\|^2,\end{aligned}$$

and this choice must be optimal, since $\|\mathbf{b}_R - \mathbf{A}\mathbf{x}\|^2 \geq 0$ for all $\mathbf{x}$.

The approximation problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$$

is solved by any $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}_R$.

One last problem remains: How do we solve $\mathbf{A}\mathbf{x} = \mathbf{b}_R$, when $\mathbf{A}$ is a rectangular matrix? In particular, we would ideally solve the least squares problem without explicitly constructing $\mathbf{b}_R$. Notice that for an optimal choice of $\mathbf{x}$ that solves $\mathbf{A}\mathbf{x} = \mathbf{b}_R$, we have

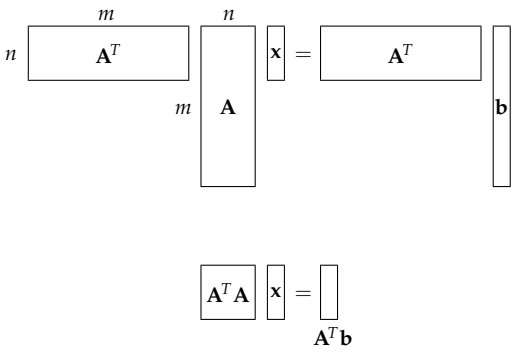
$$\mathbf{b} - \mathbf{A}\mathbf{x} = (\mathbf{b}_R - \mathbf{A}\mathbf{x}) + \mathbf{b}_N = 0 + \mathbf{b}_N.$$

Premultiply this equation by $\mathbf{A}^T$ and recall that $\mathbf{b}_N \in \mathcal{N}(\mathbf{A}^T)$ to obtain

$$\begin{aligned}\mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) &= \mathbf{A}^T\mathbf{b}_N \\ &= \mathbf{0},\end{aligned}$$

and so we rearrange to find

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}. \tag{5.18}$$



The optimal $\mathbf{x}$ that solves the least squares problem also solves the linear system (5.18), called the *normal equations*. Notice that $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n\times n}$. In many applications, the columns of $\mathbf{A}$ correspond to variables we are trying to fit, while the rows of $\mathbf{A}$ represent observations (or the results of experiments) that provide the data for the fit. We can often collect many more observations than we have variables, so $m \gg n$. Yet the system $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ only requires that we solve a small $n \times n$ system to find the best fit $\mathbf{x}$.

When does $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ have a *unique* solution? Recall that Lemma 5.2 showed $\mathcal{N}(\mathbf{A}) = \mathcal{N}(\mathbf{A}^T\mathbf{A})$, establishing the following key result.

Theorem 5.6. *The normal equations $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ have a unique solution for all $\mathbf{b} \in \mathbb{R}^m$ if and only if $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$.*

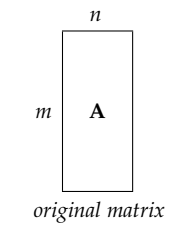
The SVD gives much deeper insight into the normal equations; We will postpone a full discussion of the least squares problem for Chapter 7. For now, let us simply consider the case where $\mathbf{A} \in \mathbb{R}^{m\times n}$ has full column rank, $\text{rank}(\mathbf{A}) = r = n$ and $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$. Thus the least squares problem has a unique solution

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}.$$

In this equation the matrix $\mathbf{A}^+ := (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \in \mathbb{R}^{n\times m}$ functions something like the inverse $\mathbf{A}^{-1}$ of a square matrix in the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ to $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Substituting the reduced SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ for $\mathbf{A}$, we can compute an expression for the pseudoinverse in terms of the SVD:

$$\begin{aligned}
\mathbf{A}^+ &= \left((\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)\right)^{-1}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T \\
&= \left(\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^{-1}\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
&= (\mathbf{V}\mathbf{\Sigma}\mathbf{\Sigma}\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \\
&= \mathbf{V}(\mathbf{\Sigma}^{-1}\mathbf{\Sigma}^{-1}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \sum_{j=1}^{r}\frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j^T.
\end{aligned}$$

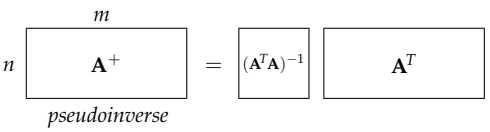


original matrix

Here we have used the fact that $\mathbf{V} \in \mathbb{R}^{n\times n}$ is an invertible matrix with orthonormal columns, so $\mathbf{V}^T = \mathbf{V}^{-1}$.

Definition 5.2. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ have rank n. The* pseudoinverse *of $\mathbf{A}$ is*

$$\mathbf{A}^+ := (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \sum_{j=1}^{r}\frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j^T.$$

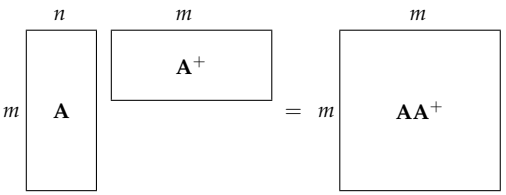


pseudoinverse

We make final observation: the matrix $\mathbf{A}\mathbf{A}^+$ has a special form:

$$\mathbf{A}\mathbf{A}^+ = \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)\left(\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\right) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \mathbf{U}\mathbf{U}^T$$

is the orthogonal projector onto $\mathcal{R}(\mathbf{U}) = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_n\} = \mathcal{R}(\mathbf{A})$.



orthogonal projector onto $\mathcal{R}(\mathbf{A})$

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ have rank n.

- The solution of the least squares problem (5.16) is then simply

$$\mathbf{x} = \mathbf{A}^{+}\mathbf{b}.$$

- The least squares approximation $\mathbf{Ax}$ to $\mathbf{b}$ is

$$\mathbf{Ax} = \mathbf{AA}^{+}\mathbf{b} = \mathbf{UU}^{T}\mathbf{b},$$

  which is the orthogonal projection (best approximation) of $\mathbf{b}$ onto $\mathcal{R}(\mathbf{U}) = \mathcal{R}(\mathbf{A})$.

- The least squares residual is

$$\mathbf{b} - \mathbf{Ax} = (\mathbf{I} - \mathbf{UU}^{T})\mathbf{b} \in \mathcal{N}(\mathbf{A}^{T}),$$

  which is orthogonal to the approximating subspace $\mathcal{R}(\mathbf{A})$.

Example 5.6. Consider the matrix introduced in Example 5.1,

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ \sqrt{2} & -\sqrt{2} \end{bmatrix} \in \mathbb{R}^{3\times 2}.$$

The pseudoinverse is

$$\mathbf{A}^{+} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U} = \sum_{j=1}^{2} \frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j = \begin{bmatrix} 1/2 & 0 & 1/(2\sqrt{2}) \\ 1/2 & 0 & -1/(2\sqrt{2}) \end{bmatrix} \in \mathbb{R}^{2\times 3},$$

while the orthogonal projector onto $\mathcal{R}(\mathbf{A})$ is

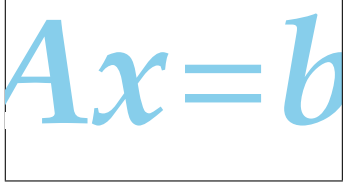
$$\mathbf{AA}^{+} = \mathbf{UU}^{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Notice that while $\mathbf{UU}^{T} \neq \mathbf{I}$, this matrix $\mathbf{UU}^{T}$ "acts like the identity" on $\mathcal{R}(\mathbf{A})$: for any $\mathbf{b} \in \mathcal{R}(\mathbf{A})$, $\mathbf{UU}^{T}\mathbf{b} = \mathbf{b}$.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 6
Matrix Approximation via the SVD

Often the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ represents a large-scale data set: for example, m samples of n distinct variables, or a digital photograph m pixels high and n pixels wide. In both these cases we might expect quite a bit of *redundancy* in the data: some variables may be highly correlated (making n large) and samples may be quite similar (making m large); a picture contains patches of nearly identical colors.

Rather than storing all $m \times n$ real numbers that make up $\mathbf{A}$, we seek a more efficient approach that reduces storage, and, *more importantly*, reveals something about the structure of our data set. This chapter is devoted to such matrix approximation problems, which inevitably reduce to applications of the SVD. Before we can consider low rank approximation, principal component analysis, and recommender systems, we must develop a basic tool: a way to measure the distance between two matrices.

6.1 Matrix norms

How 'large' is a matrix? We do not mean dimension – but how large, in aggregate, are its entries? One can imagine a multitude of ways to measure the entries; perhaps most natural is to sum the squares of the entries, then take the square root. This idea is useful, but we prefer a more subtle alternative that is of more universal utility throughout mathematics: we shall gauge the size $\mathbf{A} \in \mathbb{R}^{m\times n}$ by the maximum amount it can stretch a vector, $\mathbf{x} \in \mathbb{R}^n$. That is, we will measure $\|\mathbf{A}\|$ by the largest that $\|\mathbf{A}\mathbf{x}\|$ can be. Of course, we can inflate $\|\mathbf{A}\mathbf{x}\|$ as much as we like simply by making $\|\mathbf{x}\|$ larger, which we avoid by imposing a normalization: $\|\mathbf{x}\| = 1$. We arrive at the definition

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|.$$

First, suppose that $\mathbf{Q}$ is some matrix with orthonormal columns, so that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. Then, using the key fact on page 11,

$$\|\mathbf{Q}\mathbf{x}\|^2 = (\mathbf{Q}\mathbf{x})^T(\mathbf{Q}\mathbf{x}) = \mathbf{x}^T\mathbf{Q}^T\mathbf{Q}\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|^2,$$

so premultiplying by $\mathbf{Q}$ does not alter the norm of $\mathbf{x}$. Now substitute the full SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ for $\mathbf{A}$:

$$\|\mathbf{A}\mathbf{x}\| = \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{x}\| = \|\mathbf{\Sigma}\mathbf{V}^T\mathbf{x}\|,$$

where we have used the orthonormality of the columns of $\mathbf{U}$. Now define a new variable $\mathbf{y} = \mathbf{V}^T\mathbf{x}$ (which means $\mathbf{V}\mathbf{y} = \mathbf{x}$), and notice that $\|\mathbf{x}\| = \|\mathbf{V}^T\mathbf{x}\| = \|\mathbf{y}\|$, since $\mathbf{V}$ is a square matrix with orthonormal columns (and hence orthonormal rows). Now we can compute the matrix norm:

The fact that $\mathbf{V}$ is square and has orthonormal columns implies that both $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^T = \mathbf{I}$. This means that $\|\mathbf{V}^T\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|^2$.

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{\Sigma}\mathbf{V}^T\mathbf{x}\| = \max_{\|\mathbf{V}\mathbf{y}\|=1} \|\mathbf{\Sigma}\mathbf{y}\| = \max_{\|\mathbf{y}\|=1} \|\mathbf{\Sigma}\mathbf{y}\|$$

So the norm of $\mathbf{A}$ is the same as the norm of $\mathbf{\Sigma}$. We now must figure out how to pick the unit vector $\mathbf{y}$ to maximize $\|\mathbf{\Sigma}\mathbf{y}\|$. This is easy: we want to optimize

$$\|\mathbf{\Sigma}\mathbf{y}\|^2 = \sigma_1^2|y_1|^2 + \cdots + \sigma_r^2|y_r|^2$$

subject to $1 = \|\mathbf{y}\|^2 \geq |y_1|^2 + \cdots + |y_r|^2$. Since $\sigma_1 \geq \cdots \geq \sigma_r$,

Alternatively, you could compute $\|\mathbf{\Sigma}\|$ by maximizing $f(\mathbf{y}) = \|\mathbf{\Sigma}\mathbf{y}\|$ subject to $\|\mathbf{y}\| = 1$ using the Lagrange multiplier technique from vector calculus.

$$\begin{aligned}\|\mathbf{\Sigma}\mathbf{y}\|^2 &= \sigma_1^2|y_1|^2 + \cdots + \sigma_r^2|y_r|^2 \\ &\leq \sigma_1^2\Big(|y_1|^2 + \cdots + |y_r|^2\Big) \leq \sigma_1^2\|\mathbf{y}\|^2 = \sigma_1^2,\end{aligned}$$

resulting in the upper bound

$$\|\mathbf{\Sigma}\| = \max_{\|\mathbf{y}\|=1} \|\mathbf{\Sigma}\mathbf{y}\| \leq \sigma_1. \tag{6.1}$$

Will any unit vector $\mathbf{y}$ attain this upper bound? That is, can we find such a vector so that $\|\mathbf{\Sigma}\mathbf{y}\| = \sigma_1$? Sure: just take $\mathbf{y} = [1, 0, \cdots, 0]^T$ to be the first column of the identity matrix. For this special vector,

$$\|\mathbf{\Sigma}\mathbf{y}\|^2 = \sigma_1^2|y_1|^2 + \cdots + \sigma_r^2|y_r|^2 = \sigma_1^2.$$

Since $\|\mathbf{\Sigma}\mathbf{y}\|$ can be no larger than σ_1 for any $\mathbf{y}$, and since $\|\mathbf{\Sigma}\mathbf{y}\| = \sigma_1$ for at least one choice of $\mathbf{y}$, we conclude

$$\|\mathbf{\Sigma}\| = \max_{\|\mathbf{y}\|=1} \|\mathbf{\Sigma}\mathbf{y}\| = \sigma_1,$$

and hence *the norm of a matrix is its largest singular value*:

$$\|\mathbf{A}\| = \sigma_1.$$

Consider the matrix

$$\mathbf{A} = \begin{bmatrix} 1/2 & 1 \\ -1/2 & 1 \end{bmatrix} = \left(\frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2}/2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^T.$$

We see from this SVD that $\|\mathbf{A}\| = \sigma_1 = \sqrt{2}$. For this example the vector $\mathbf{Ax}$ has the form

$$\begin{aligned} \mathbf{Ax} &= \sigma_1(\mathbf{v}_1^T\mathbf{x})\mathbf{u}_1 + \sigma_2(\mathbf{v}_2^T\mathbf{x})\mathbf{u}_2 \\ &= \sqrt{2}x_2\mathbf{u}_1 + \frac{\sqrt{2}}{2}x_1\mathbf{u}_2, \end{aligned}$$

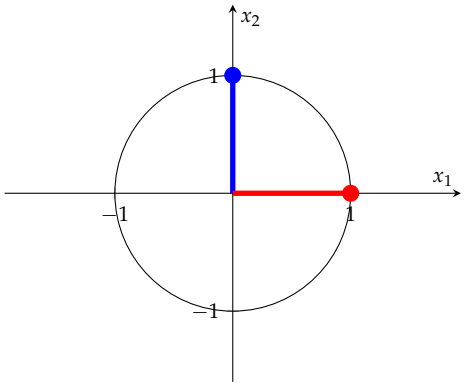
so $\mathbf{Ax}$ is a blend of some expansion in the $\mathbf{u}_1$ direction and some contraction in the $\mathbf{u}_2$ direction. We maximize the size of $\mathbf{Ax}$ by picking an $\mathbf{x}$ for which $\mathbf{Ax}$ is maximally rich in $\mathbf{u}_1$, i.e., $\mathbf{x} = \mathbf{v}_1$.

In Python, you can compute the matrix norm directly using Numpy's `norm` command:

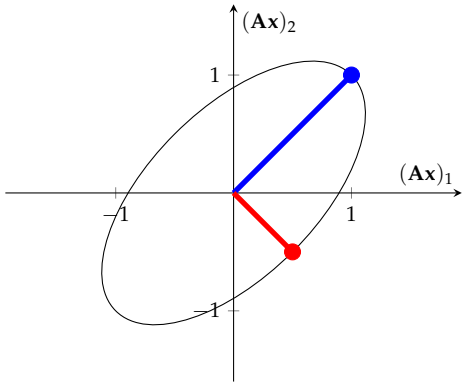
```
normA = np.linalg.norm(A,2)
```

Here the `2` in the second argument is crucial: there are other ways to define the norm of the matrix, and the `2` tells Python to use the definition that we have just described. Of course, you could also compute the norm by computing the SVD and taking the maximum singular value:

```
U, S, Vt = np.linalg.svd(A)
normA = np.max(S)
```



Every unit vector $\mathbf{x}$ in $\mathbb{R}^2$ is a point where $\|\mathbf{x}\|^2 = x_1^2 + x_2^2 = 1$, so the set of all such vectors traces out the *unit circle* shown in black in the plot above. We highlight two distinguished vectors: $\mathbf{x} = \mathbf{v}_1$ (blue) and $\mathbf{x} = \mathbf{v}_2$ (red).



The plot above shows $\mathbf{Ax}$ for all unit vectors $\mathbf{x}$, which traces out an ellipse in $\mathbb{R}^2$. The vector $\mathbf{x} = \mathbf{v}_1$ is mapped to $\mathbf{Ax} = \sigma_1\mathbf{u}_1$ (blue), and this is the most $\mathbf{A}$ stretches any unit vector; $\mathbf{x} = \mathbf{v}_2$ is mapped to $\mathbf{Ax} = \sigma_2\mathbf{u}_2$ (red), which gives the smallest value of $\|\mathbf{Ax}\|$.

6.2 *Low-rank approximation*

Perhaps the most important property of the singular value decomposition is its ability to immediately deliver optimal low-rank approximations to a matrix. The dyadic form

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$$

writes the rank-r matrix $\mathbf{A}$ as the sum of the r rank-1 matrices

$$\sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

Since $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, we might hope that the partial sum

$$\sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$$

will give a good approximation to $\mathbf{A}$ for some value of k that is *much* smaller than r (mathematicians write $k \ll r$ for emphasis). This is

especially true in situations where $\mathbf{A}$ models some low-rank phenomenon, but some noise (such as random sampling errors, when the entries of $\mathbf{A}$ are measured from some physical process) causes $\mathbf{A}$ to have much larger rank. If the noise is small relative to the "true" data in $\mathbf{A}$, we expect $\mathbf{A}$ to have a number of very small singular values that we might wish to neglect as we work with $\mathbf{A}$. We will see examples of this kind of behavior in the next chapter.

For square diagonalizable matrices,

$$\mathbf{A} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^{-1},$$

the eigenvalue decomposition can also lead to an expression for $\mathbf{A}$ as the sum of rank-1 matrices:

$$\mathbf{A} = \sum_{j=1}^{n} \lambda_j \mathbf{w}_j \widehat{\mathbf{w}}_j^T,$$

where $\widehat{\mathbf{w}}_j^T$ denotes the jth row of $\mathbf{W}^{-1}$.

Note that if $\mathbf{A}$ has real entries, then the SVD will only have real entries. This is not generally the case for the eigenvalue decomposition when $\mathbf{A}$ is a nonsymmetric matrix.

Three key distinctions make the singular value decomposition a better tool for developing low-rank approximations to $\mathbf{A}$.

1. The SVD holds for all rectangular matrices, while the eigenvalue decomposition only holds for square matrices that are diagonalizable.

2. The singular values are nonnegative real numbers whose ordering

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

   gives a natural way to understand how much the rank-1 matrices $\sigma_j \mathbf{u}_j \mathbf{v}_j^T$ contribute to $\mathbf{A}$. In contrast, the eigenvalues will generally be complex numbers, and thus lack the same natural order; it is harder to understand the significance of each rank-1 matrix $\lambda_j \mathbf{w}_j \widehat{\mathbf{w}}_j^T$.

3. The eigenvectors are not generally orthogonal, and this can skew the rank-1 matrices $\lambda_j \mathbf{w}_j \widehat{\mathbf{w}}_j^T$ away from giving good approximations to $\mathbf{A}$. In particular, we can find that $\|\mathbf{w}_j \widehat{\mathbf{w}}_j^T\| \gg 1$, whereas the matrices $\mathbf{u}_j \mathbf{v}_j^T$ from the SVD always satisfy $\|\mathbf{u}_j \mathbf{v}_j^T\| = 1$.

This last point is subtle, so let us investigate it with an example. Consider

$$\mathbf{A} = \begin{bmatrix} 2 & 100 \\ 0 & 1 \end{bmatrix}$$

with eigenvalues $\lambda_1 = 2$ and $\lambda_2 = 1$ and eigenvalue decomposition

$$\mathbf{A} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^{-1} = \begin{bmatrix} 1 & 1 \\ 0 & -1/100 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 100 \\ 0 & -100 \end{bmatrix}$$

$$
\begin{aligned}
&= \lambda_1 \mathbf{w}_1 \widehat{\mathbf{w}}_1^T + \lambda_2 \mathbf{w}_2 \widehat{\mathbf{w}}_2^T \\
&= 2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 100 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ -1/100 \end{bmatrix} \begin{bmatrix} 0 & -100 \end{bmatrix} \\
&= 2 \begin{bmatrix} 1 & 100 \\ 0 & 0 \end{bmatrix} + 1 \begin{bmatrix} 0 & -100 \\ 0 & 1 \end{bmatrix}.
\end{aligned}
$$

Let us inspect individually the two rank-1 matrices that appear in the eigendecomposition:

$$
\lambda_1 \mathbf{w}_1 \widehat{\mathbf{w}}_1^T = \begin{bmatrix} 2 & 200 \\ 0 & 0 \end{bmatrix}, \qquad \lambda_2 \mathbf{w}_2 \widehat{\mathbf{w}}_2^T = \begin{bmatrix} 0 & -100 \\ 0 & 1 \end{bmatrix}.
$$

Neither matrix individually gives a good approximation to $\mathbf{A}$*:*

$$
\mathbf{A} - \lambda_1 \mathbf{w}_1 \widehat{\mathbf{w}}_1^T = \begin{bmatrix} 0 & -100 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{A} - \lambda_2 \mathbf{w}_2 \widehat{\mathbf{w}}_2^T = \begin{bmatrix} 2 & 200 \\ 0 & 0 \end{bmatrix}.
$$

Both rank-1 "approximations" to $\mathbf{A}$ leave large errors!

Contrast this situation with the rank-1 approximation $\sigma_1 \mathbf{u}_1 \mathbf{v}_1^T$ given by the SVD for this $\mathbf{A}$. To five decimal digits, we have

$$
\begin{aligned}
\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T &= \begin{bmatrix} 0.99995 & -0.01000 \\ 0.01000 & 0.99995 \end{bmatrix} \begin{bmatrix} 100.025 & 0 \\ 0 & 0.020 \end{bmatrix} \begin{bmatrix} 0.01999 & 0.99980 \\ -0.99980 & 0.01999 \end{bmatrix} \\
&= \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ +\sigma_2 \mathbf{u}_2 \mathbf{v}_2^T \\
&= 100.025 \begin{bmatrix} 0.99995 \\ 0.01000 \end{bmatrix} \begin{bmatrix} 0.01999 & 0.99980 \end{bmatrix} + 0.020 \begin{bmatrix} -0.01000 \\ 0.99995 \end{bmatrix} \begin{bmatrix} -0.99980 & 0.01999 \end{bmatrix} \\
&= 100.025 \begin{bmatrix} 0.01999 & 0.99975 \\ 0.00020 & 00.00999 \end{bmatrix} + 0.020 \begin{bmatrix} 0.00999 & -0.00020 \\ -.99975 & 0.01999 \end{bmatrix}.
\end{aligned}
$$

Like the eigendecomposition, the SVD breaks $\mathbf{A}$ into two rank-1 pieces:

$$
\sigma_1 \mathbf{u}_1 \mathbf{v}_1^T = \begin{bmatrix} 1.99980 & 100.00000 \\ 0.01999 & 0.99960 \end{bmatrix}, \qquad \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T = \begin{bmatrix} 0.00020 & 0.00000 \\ -0.01999 & 0.00040 \end{bmatrix}.
$$

The first of these, the dominant term in the SVD, gives an *excellent* approximation to $\mathbf{A}$:

$$
\mathbf{A} - \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T = \begin{bmatrix} 0.00020 & 0.00000 \\ -0.01999 & 0.00040 \end{bmatrix}.
$$

The key factor making this approximation so good is that $\sigma_1 \gg \sigma_2$. What is more remarkable is that the dominant part of the singular value decomposition is actually the *best* low-rank approximation for all matrices.

Definition 6.1. *Let $\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$ be a rank-r matrix, written in terms of its singular value decomposition. Then for any $k \leq r$, the* truncated singular value decomposition *of rank-k is the partial sum*

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

Theorem 6.1 (Schmidt–Mirsky–Eckart–Young). *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. Then for all $k \leq \text{rank}(\mathbf{A})$, the truncated singular value decomposition*

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$$

is a best rank-k approximation to $\mathbf{A}$, in the sense that

$$\sigma_{k+1} = \|\mathbf{A} - \mathbf{A}_k\| = \min_{\text{rank}(\mathbf{X}) \leq k} \|\mathbf{A} - \mathbf{X}\|.$$

It is easy to see that this $\mathbf{A}_k$ gives the approximation error σ_{k+1}, since

$$\mathbf{A} - \mathbf{A}_k = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T - \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T = \sum_{j=k+1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T,$$

and this last expression is an SVD for the error in the approximation $\mathbf{A} - \mathbf{A}_k$. As described in Section 6.1, the norm of a matrix equals its largest singular value, so

$$\|\mathbf{A} - \mathbf{A}_k\| = \left\| \sum_{j=k+1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_k^T \right\| = \sigma_{k+1}.$$

To complete the proof, one needs to show that no other rank-k matrix can come closer to $\mathbf{A}$ than $\mathbf{A}_k$. This pretty argument is a bit too intricate for this course, but we include it in the margin for those that are interested.

You can construct low-rank approximations in Python in several different ways. Suppose that the rank `k` of the approximation has been defined. Then the following commands build up the best approximation from the dyadic form, using the `np.outer` command for the outer product.

```
U, S, Vt = np.linalg.svd(A)
Ak = np.zeros(A.shape)
for j in range(0,k):
    Ak = Ak + S[j]*np.outer(U[:,j],Vt[j,:])
```

Alternatively, one can skip the loop and build $\mathbf{A}_k$ via the reduced form of its SVD.

Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be any rank-k matrix. The Fundamental Theorem of Linear Algebra gives $\mathbb{R}^n = \mathcal{R}(\mathbf{X}^T) \oplus \mathcal{N}(\mathbf{X})$. Since $\text{rank}(\mathbf{X}^T) = \text{rank}(\mathbf{X}) = k$, notice that $\dim(\mathcal{N}(\mathbf{X})) = n - k$. From the singular value decomposition of $\mathbf{A}$ extract $\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}$, a basis for some $k+1$ dimensional subspace of $\mathbb{R}^n$. Since $\mathcal{N}(\mathbf{X}) \subseteq \mathbb{R}^n$ has dimension $n - k$, it must be that the intersection

$$\mathcal{N}(\mathbf{X}) \cap \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$$

has dimension at least one. (Otherwise, $\mathcal{N}(\mathbf{X}) \oplus \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$ would be an $n+1$ dimensional subspace of $\mathbb{R}^n$: impossible!) Let $\mathbf{z}$ be some unit vector in that intersection: $\|\mathbf{z}\| = 1$ and

$$\mathbf{z} \in \mathcal{N}(\mathbf{X}) \cap \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}.$$

Expand $\mathbf{z} = \gamma_1 \mathbf{v}_1 + \cdots + \gamma_{k+1} \mathbf{v}_{k+1}$, so that $\|\mathbf{z}\| = 1$ implies

$$1 = \mathbf{z}^T \mathbf{z} = \left(\sum_{j=1}^{k+1} \gamma_j \mathbf{v}_j \right)^* \left(\sum_{j=1}^{k+1} \gamma_j \mathbf{v}_j \right) = \sum_{j=1}^{k+1} |\gamma_j|^2.$$

Since $\mathbf{z} \in \mathcal{N}(\mathbf{X})$, we have

$$\|\mathbf{A} - \mathbf{X}\| \geq \|(\mathbf{A} - \mathbf{X})\mathbf{z}\| = \|\mathbf{A}\mathbf{z}\|,$$

and then

$$\|\mathbf{A}\mathbf{z}\| = \left\| \sum_{j=1}^{k+1} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \mathbf{z} \right\| = \left\| \sum_{j=1}^{k+1} \sigma_j \gamma_j \mathbf{u}_j \right\|.$$

Since $\sigma_{k+1} \leq \sigma_k \leq \cdots \leq \sigma_1$ and the $\mathbf{u}_j$ vectors are orthogonal,

$$\left\| \sum_{j=1}^{k+1} \sigma_j \gamma_j \mathbf{u}_j \right\|_2 \geq \sigma_{k+1} \left\| \sum_{j=1}^{k+1} \gamma_j \mathbf{u}_j \right\|_2.$$

But notice that

$$\left\| \sum_{j=1}^{k+1} \gamma_j \mathbf{u}_j \right\|_2^2 = \sum_{j=1}^{k+1} |\gamma_j|^2 = 1,$$

where the last equality was derived above from the fact that $\|\mathbf{z}\|_2 = 1$. In conclusion, for any rank-k matrix $\mathbf{X}$,

$$\|\mathbf{A} - \mathbf{X}\|_2 \geq \sigma_{k+1} \left\| \sum_{j=1}^{k+1} \gamma_j \mathbf{u}_j \right\|_2 = \sigma_{k+1}.$$

(This proof is adapted from §3.2.3 of Demmel's text.)

James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997

```
Ak = U[:,0:k]@np.diag(S[0:k])@Vt[0:k,:]
```

or the slightly more efficient form

```
Ak = (U[:,0:k]*S[0:k])@Vt[0:k,:]
```

6.2.1 *Compressing images with low rank approximations*

Image compression provides the most visually appealing application of the low-rank matrix factorization ideas we have just described. An image can be represented as a matrix. For example, typical grayscale images consist of a rectangular array of pixels, m in the vertical direction, n in the horizontal direction. The color of each of those pixels is denoted by a single number, an integer between 0 (black) and 255 (white). (This gives $2^8 = 256$ different shades of gray for each pixel. Color images are represented by three such matrices: one for red, one for green, and one for blue. Thus each pixel in a typical color image takes $(2^8)^3 = 2^{24} = 16,777,216$ shades.)

Images are ripe for data compression: Often they contain broad regions of similar colors, and in many areas of the image adjacent rows (or columns) will look quite similar. If the image stored in $\mathbf{A}$ can be represented well by a rank-k matrix, then one can approximate $\mathbf{A}$ by storing only the leading k singular values and vectors. To build this approximation

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T,$$

one need only store $k(1 + m + n)$ values. When $k(1 + m + n) \ll mn$, there will be a significant savings in storage, thus giving an effective compression of $\mathbf{A}$.

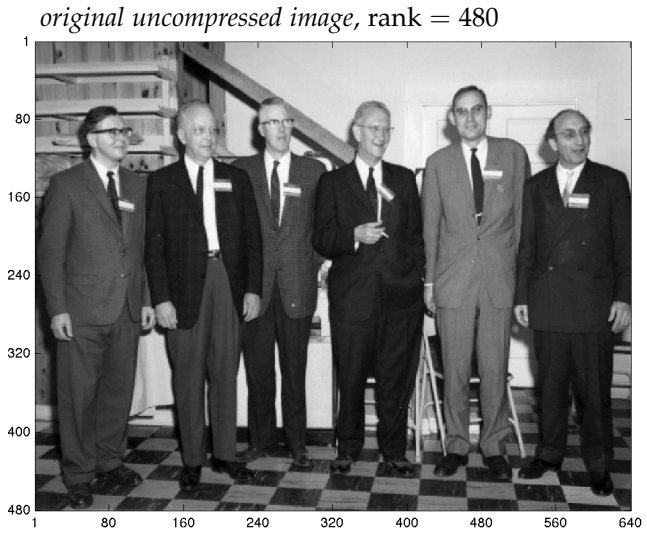


Figure 6.1: A sample image: the founders of numerical linear algebra at an early Gatlinburg Symposium. From left to right: Jim Wilkinson, Wallace Givens, George Forsythe, Alston Householder, Peter Henrici, and Friedrich Bauer. This image is built into MATLAB; access it via `load gatlin`.

Let us look at an example to see how effective this image compression can be. The image in Figure 6.1 shows some of the key developers of the numerical linear algebra algorithms we have studied this semester, gathered in Gatlinburg, Tennessee, for an important early conference in the field. The image is of size 480×640, so $\text{rank}(\mathbf{A}) \leq 480$. We shall compress this image with truncated singular value decompositions. Figures 6.3 and 6.4 show compressions of $\mathbf{A}$ for dimensions ranging from $k = 200$ down to $k = 1$. For $k = 200$ and 100, the compression $\mathbf{A}_k$ provides an excellent proxy for the full image $\mathbf{A}$. For $k = 50$, 25 and 10, the quality degrades a bit, but even for $k = 10$ you can still tell that the image shows six men in suits standing on a patterned floor. For $k \leq 5$ we lose much of the quality, but isn't it remarkable how much structure is still apparent even when $k = 5$? The last image is interesting as a visualization of a rank-1 matrix: each row is a multiple of all the other rows, and each column is a multiple of all the other columns.

We gain an understanding of the quality of this compression by looking at the singular values of $\mathbf{A}$, shown in Figure 6.2. The first singular value σ_1 is a about an order of magnitude larger than the rest, and the singular values decay quite rapidly. (Notice the logarithmic vertical axis.) We have $\sigma_1 \approx 15,462$, while $\sigma_{50} \approx 204.48$. When we truncate the singular value decomposition at $k = 50$, the neglected terms in the singular value decomposition do not make a major contribution to the image.

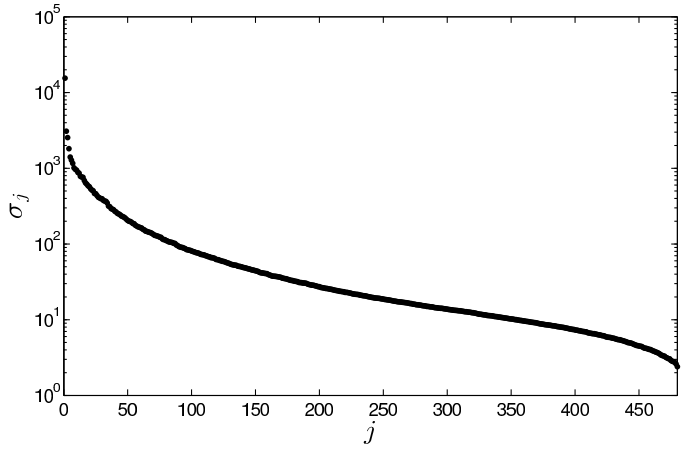
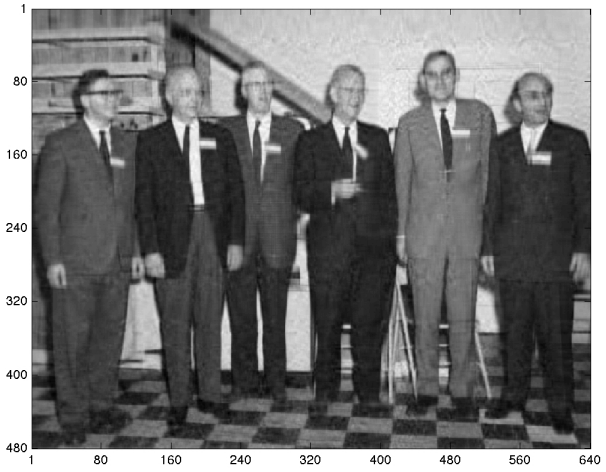


Figure 6.2: Singular values of the 480×640 Gatlinburg image matrix. The first few singular values are much larger than the rest, suggesting the potential for accurate low-rank approximation (compression).

truncated SVD, rank $k = 200$

truncated SVD, rank $k = 100$

truncated SVD, rank $k = 50$



truncated SVD, rank $k = 25$

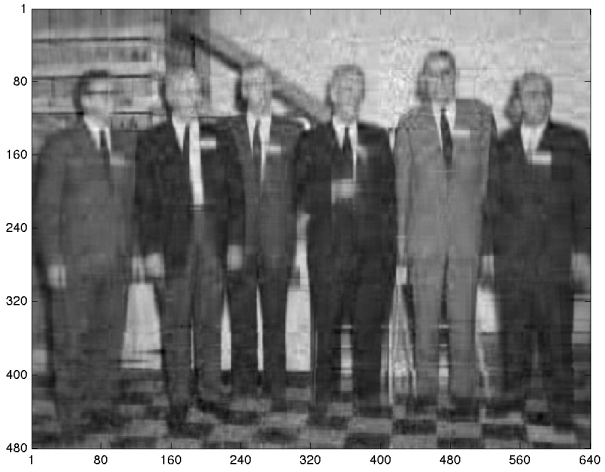


Figure 6.3: Compressions of the Gatlinburg image in Figure 6.1 using truncated SVDs $\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$. Each of these images can be stored with less memory than the original full image. The rank-25 image could be be useful as a "thumbnail" sketch of the image (e.g., an icon on a computer desktop).

To investigate this low-rank approximation a little more deeply, let us introduce another image, a carved grotesque, shown in Figure 6.5. This grayscale image comprises 644×500 pixels, suggesting that $\text{rank}(\mathbf{A}) = 500$. Figure 6.6 shows that singular values decay much like those for the Gatlinburg matrix (Figure 6.2); indeed, the grotesque's first singular value is at least ten times larger than all the others, with $\sigma_1 \approx 8.84 \times 10^4$ while $\sigma_2 \approx 7.91 \times 10^3$.

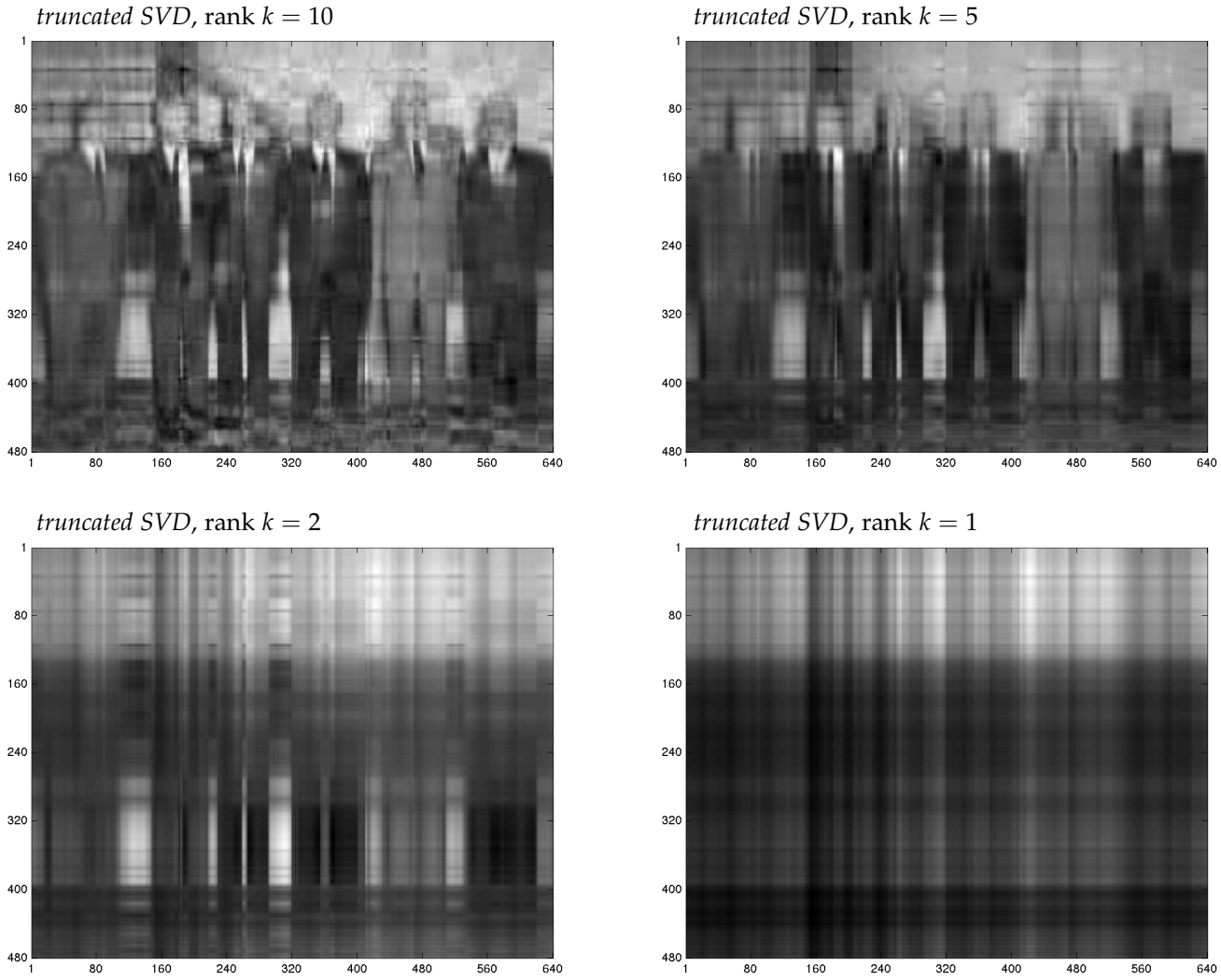


Figure 6.4: Continuation of Figure 6.4, showing compressions of rank 10, 5, 2, and 1. Note the striping characteristic of low-rank structure.

Based on these singular values, we expect a strong low-rank approximation. Figure 6.7 shows rank-k truncated SVD approximations for eight values of k. Indeed, given the dominant size of σ_1, we see the major structure of the frame and border evident even in the $k = 1$ compression.

To emphasize how the individual components $\sigma_j \mathbf{u}_j \mathbf{v}^T$ contribute to the sum

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}^T, \tag{6.2}$$

we shall take a closer look at the image in Figure 6.5. To make this point as clearly as possible, we introduce a new color map that shows positive values in blue and negative values in red, as illustrated in

Figure 6.5: A grotesque carved in a door of the 16th century Church of Santa Croce, Riva San Vitale, Switzerland.

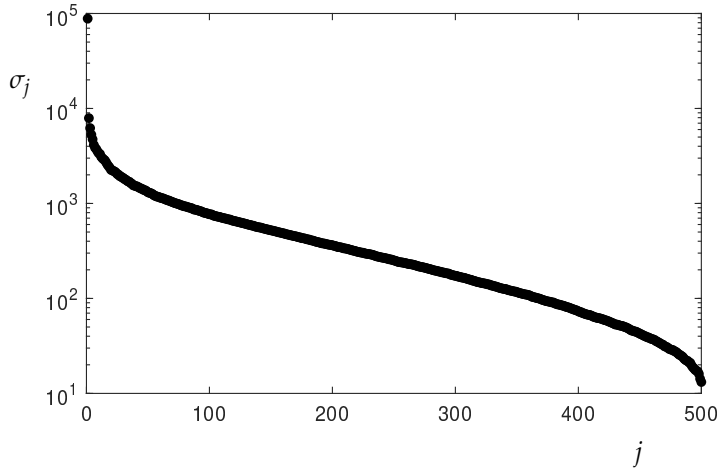


Figure 6.6: Singular values of the 644×500 Santa Croce grotesque. The first singular value is ten times or more larger than the others.

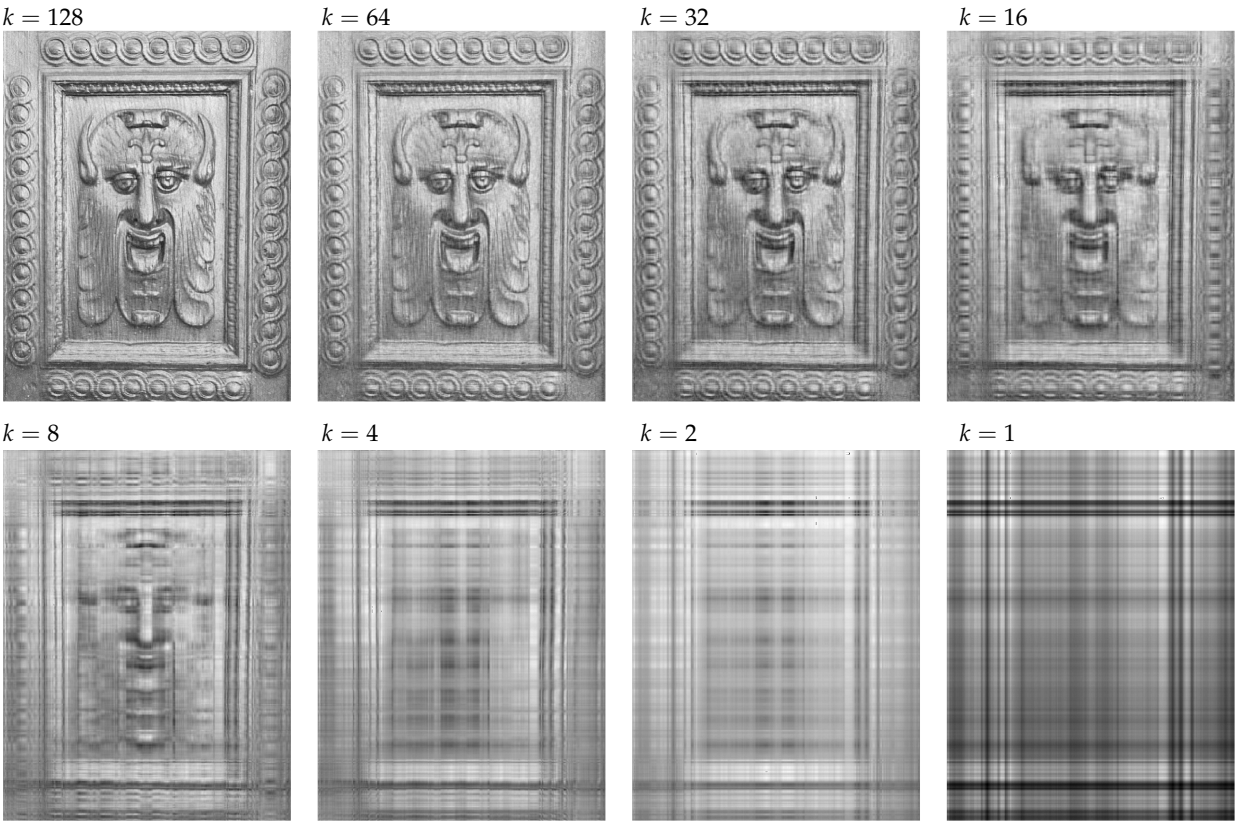


Figure 6.7: Rank-k truncated SVD compressed versions of the Santa Croce grotesque. (Close examination of the original image shows numerous worm holes, especially in the lower left of the panel. Notice how these are reduced when $k = 128$ and essentially disappear when $k = 64$.)

Figure 6.8. (The image $\mathbf{A}$ has integer entries between $[0, 255]$, but the truncated SVDs $\mathbf{A}_k$ need not have integer entries, and the individual terms $\sigma_j \mathbf{u}_j \mathbf{v}_j^T$ can have negative entries.)

Figure 6.9 shows how this image is assembled from the individual terms in the dyadic form of the singular value decomposition (6.2).

Since $\sigma_1 \approx 8.84 \times 10^4$ is so much larger than $\sigma_2 \approx 7.91 \times 10^3$, the first term dominates (hence the dark blue color): the interior frame around the face is already evident. The subsequent matrices $\sigma_j \mathbf{u}_j \mathbf{v}_j^T$ for $j \geq 2$ add more modest corrections that fill in details of the image. Some of these effects can be readily picked out: for example, $\sigma_3 \mathbf{u}_3 \mathbf{v}_3^T$ adjusts for the row of carving at the top of the image; $\sigma_5 \mathbf{u}_5 \mathbf{v}_5^T$ fills in the grotesque's nose. Since the singular values σ_j are decreasing as j grows, these terms make smaller and smaller contributions.

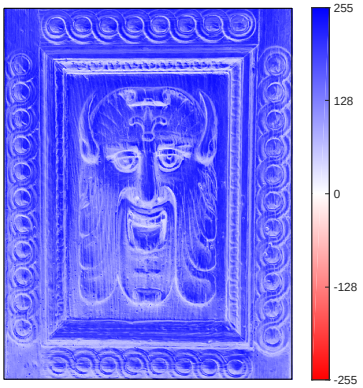


Figure 6.8: The Santa Croce grotesque with an extended color map to highlight negative values (red).

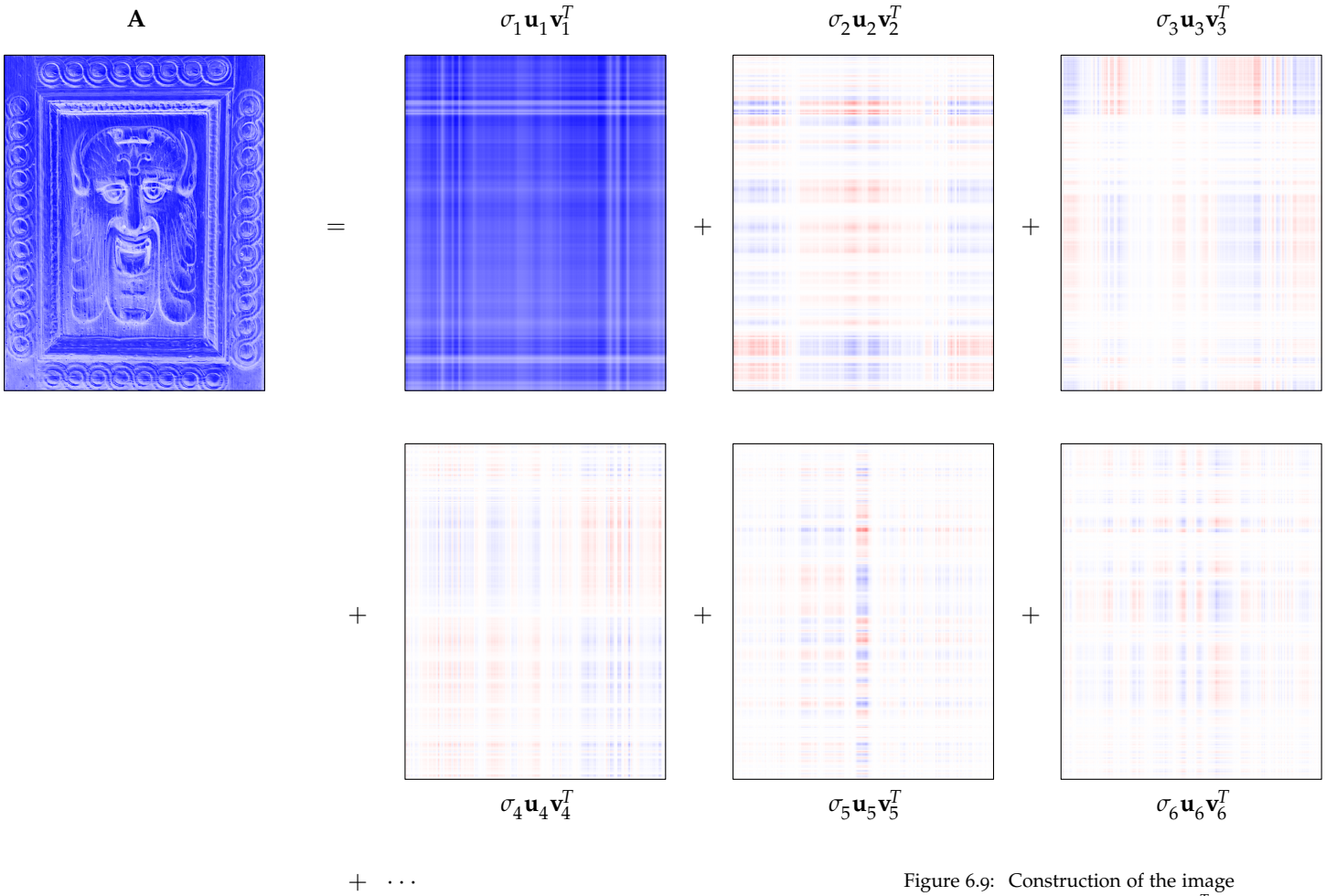


Figure 6.9: Construction of the image $\mathbf{A}$ from the individual terms $\sigma_j \mathbf{u}_j \mathbf{v}_j^T$. The first term makes a major contribution; each subsequent term makes a small adjustment, and these adjustments diminish in significance as j increases.

6.3 Principal Component Analysis

Linear algebra enables the analysis of the volumes of data that now so commonly arise from applications ranging from basic science to public policy. Such measured data often depends on many factors, and we seek to identify those that are most critical. Within this realm of multivariate statistics, *principal component analysis* (PCA) is a fundamental tool.

Linear algebraists often say, "PCA is the SVD" – in this section, we will explain what this means, and some of the subtleties involved.

6.3.1 Variance and covariance

To understand principal component analysis, we need some basic notions from statistics, described in any basic textbook. For a general description of PCA along with numerous applications, see the text by Jolliffe [1], whose presentation shaped parts of our discussion here.

[1] I. T. Jolliffe. *Principal Component Analysis*. Springer, New York, second edition, 2002

The *expected value*, or *mean*, of a random variable X is denoted $\mathsf{E}[X]$. The expected value is a *linear operation*, meaning that for any constant $\alpha \in \mathbb{R}$ and random variables X and Y, we have

$$\mathsf{E}[\alpha X + Y] = \alpha \mathsf{E}[X] + \mathsf{E}[Y].$$

Further, note that for the constant $\alpha \in \mathbb{R}$, $\mathsf{E}[\alpha] = \alpha$.

The *variance* of X describes how much X is expected to deviate from its mean,

$$\mathsf{Var}(X) = \mathsf{E}[(X - \mathsf{E}[X])^2],$$

which, using linearity of the expected value, takes the equivalent form

$$\mathsf{Var}(X) = \mathsf{E}[X^2] - \mathsf{E}[X]^2.$$

The *covariance* between two (potentially correlated) random variables X and Y is

$$\begin{aligned} \mathsf{Cov}(X, Y) &= \mathsf{E}[(X - \mathsf{E}[X])(Y - \mathsf{E}[Y])] \\ &= \mathsf{E}[XY] - \mathsf{E}[X]\mathsf{E}[Y]. \end{aligned}$$

with $\mathsf{Cov}(X, X) = \mathsf{Var}(X)$. These definitions of variance and covariance are the bedrock concepts underneath PCA, for with them we can understand the variance present in a linear combination of several random variables.

Suppose we have a set of real-valued random variables $X_1, \ldots, X_n$ in which we suspect there may be some redundancy. Perhaps some of these variables can be expressed as linear combinations of the others – either exactly, or nearly so. At the other extreme, there may be some way to combine $X_1, \ldots, X_n$ that captures much of the variance

in one (or a few) aggregate random variables. In particular, we shall seek scalars $\gamma_1, \ldots, \gamma_n$ such that

$$\sum_{j=1}^{n} \gamma_j X_j$$

has the largest possible variance. The definitions of variance and covariance, along with the linearity of the expected value, lead to a formula for the variance of a linear combination of random variables:

$$\begin{aligned}
\mathsf{Var}\Big(\sum_{j=1}^{n} \gamma_j X_j\Big) &= \mathsf{E}\left[\Big(\sum_{j=1}^{n} \gamma_j X_j\Big)^2 - \mathsf{E}\Big[\sum_{j=1}^{n} \gamma_j X_j\Big]^2\right] \\
&= \mathsf{E}\left[\Big(\sum_{j=1}^{n} \gamma_j X_j\Big)^2 - \Big(\sum_{j=1}^{n} \gamma_j \mathsf{E}[X_j]\Big)^2\right] \\
&= \mathsf{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} \gamma_j \gamma_k X_j X_k - \sum_{j=1}^{n}\sum_{k=1}^{n} \gamma_j \gamma_k \mathsf{E}[X_j]\mathsf{E}[X_k]\right] \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} \gamma_j \gamma_k \, \mathsf{E}\Big[X_j X_k - \mathsf{E}[X_j]\mathsf{E}[X_k]\Big] \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} \gamma_j \gamma_k \, \mathsf{Cov}(X_j, X_k). \qquad (6.3)
\end{aligned}$$

You have seen double sums like this before. If we define the *covariance matrix* $\mathbf{C} \in \mathbb{R}^{n\times n}$ having (j,k) entry

$$c_{j,k} = \mathsf{Cov}(X_j, X_k),$$

and let $\mathbf{v} = [\gamma_1, \ldots, \gamma_n]^T$, then the variance of the combined variable can be written in a beautifully compact way:

$$\mathsf{Var}\Big(\sum_{j=1}^{n} \gamma_j X_j\Big) = \mathbf{v}^T \mathbf{C} \mathbf{v}.$$

Statistics texts typically denote the covariance matrix by $\mathbf{\Sigma}$. We deviate from that notation, as it would clash with the matrix of singular values – which will soon make an appearance in this discussion.

Since the covariance function is symmetric: $\mathsf{Cov}(X,Y) = \mathsf{Cov}(Y,X)$, the matrix $\mathbf{C}$ is symmetric; it is also positive semidefinite. Why? Variance, by its definition as the expected value of the square of a real random variable, is always nonnegative. Thus the formula (6.3), which derives from the linearity of the expected value, ensures that $\mathbf{v}^T\mathbf{C}\mathbf{v} \geq 0$. (Under what circumstances can this quantity be zero?)

We can write $\mathbf{C}$ in another convenient way. Collect the random variables into the vector

$$\mathbf{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} \in \mathbb{R}^n.$$

Then the (j,k) entry of $\mathsf{E}[\mathbf{X}\mathbf{X}^T] - \mathsf{E}[\mathbf{X}]\mathsf{E}[\mathbf{X}]^T$ is

$$\mathsf{E}[X_j X_k] - \mathsf{E}[X_j]\mathsf{E}[X_k] = \mathsf{Cov}(X_j, X_k) = c_{j,k},$$

and so

$$\mathbf{C} = \mathsf{E}[\mathbf{X}\mathbf{X}^T] - \mathsf{E}[\mathbf{X}]\mathsf{E}[\mathbf{X}]^T.$$

6.3.2 *Derived variables that maximize variance*

Return now to the problem of *maximizing* the variance of $\mathbf{v}^T\mathbf{C}\mathbf{v}$. Without constraint on $\mathbf{v}$, this quantity can be arbitrarily large (assuming $\mathbf{C}$ is nonzero); thus we shall require that $\sum_{j=1}^{k} \gamma_j^2 = \|\mathbf{v}\|^2 = 1$. With this normalization, Theorem 4.4 immediately shows us how to maximize the variance $\mathbf{v}^T\mathbf{C}\mathbf{v}$: $\mathbf{v}$ should be a unit eigenvector associated with the largest magnitude eigenvalue of $\mathbf{C}$; call this vector $\mathbf{v}_1$. The associated variance, of course, is the largest eigenvalue of $\mathbf{C}$; call it

$$\lambda_1 = \mathbf{v}_1^T\mathbf{C}\mathbf{v}_1 = \max_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{C}\mathbf{v}.$$

The eigenvector $\mathbf{v}_1$ encodes the way to combine $X_1, \ldots, X_n$ to maximize variance. The new variable – *the leading principal component* – is

$$\mathbf{v}_1^T\mathbf{X} = \sum_{j=1}^{n} \gamma_j X_j.$$

You are already suspecting that a unit eigenvector associated with the second largest eigenvalue, $\mathbf{v}_2$ with $\lambda_2 = \mathbf{v}_2^T\mathbf{C}\mathbf{v}_2$, must encode the second-largest way to maximize variance.

Let us explore this intuition. To find the second-best way to combine the variables, we should insist that the next new variable, for now call it $\mathbf{w}^T\mathbf{X}$, should be *uncorrelated* with the first, i.e.,

Recall that $\mathbf{X} \in \mathbb{R}^n$ here, so $\mathbf{w}^T\mathbf{X} \in \mathbb{R}$ is a *scalar*.

$$\mathsf{Cov}(\mathbf{w}^T\mathbf{X}, \mathbf{v}_1^T\mathbf{X}) = 0.$$

However, using linearity of expectation and the fact that, e.g., $\mathbf{w}^T\mathbf{X} = \mathbf{X}^T\mathbf{w}$ for real vectors,

$$\begin{aligned}
\mathsf{Cov}(\mathbf{w}^T\mathbf{X}, \mathbf{v}_1^T\mathbf{X}) &= \mathsf{E}[(\mathbf{w}^T\mathbf{X})(\mathbf{v}_1^T\mathbf{X})] - \mathsf{E}[\mathbf{w}^T\mathbf{X}]\mathsf{E}[\mathbf{v}_1^T\mathbf{X}] \\
&= \mathsf{E}[(\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{v}_1) - \mathsf{E}[\mathbf{w}^T\mathbf{X}]\mathsf{E}[\mathbf{X}^T\mathbf{v}_1] \\
&= \mathbf{w}^T\mathsf{E}[\mathbf{X}\mathbf{X}^T]\mathbf{v}_1 - \mathbf{w}^T\mathsf{E}[\mathbf{X}]\mathsf{E}[\mathbf{X}]^T\mathbf{v}_1 && \text{Note that } \mathsf{E}[\mathbf{X}^T] = \mathsf{E}[\mathbf{X}]^T. \\
&= \mathbf{w}^T\big(\mathsf{E}[\mathbf{X}\mathbf{X}^T] - \mathsf{E}[\mathbf{X}]\mathsf{E}[\mathbf{X}]^T\big)\mathbf{v}_1 \\
&= \mathbf{w}^T\mathbf{C}\mathbf{v}_1 = \lambda_1\mathbf{w}^T\mathbf{v}_1. && \mathbf{C}\mathbf{v}_1 = \lambda_1\mathbf{v}_1.
\end{aligned}$$

Hence (assuming $\lambda_1 \neq 0$), for the combined variables $\mathbf{v}_1^T\mathbf{X}$ and $\mathbf{w}^T\mathbf{X}$ to be uncorrelated, the vectors $\mathbf{v}_1$ and $\mathbf{w}$ must be *orthogonal*, perfectly confirming your intuition: the second-best way to combine the variables is to pick $\mathbf{w}$ to be a unit eigenvector $\mathbf{v}_2$ of $\mathbf{C}$ corresponding to

the second largest eigenvalue. Since the eigenvectors of a symmetric matrix are orthogonal, we optimize over all vectors orthogonal to $\mathbf{u}_1$. The associated variance of $\mathbf{v}_2^T\mathbf{X}$ is thus

$$\lambda_2 = \max_{\substack{\|\mathbf{w}\|=1 \\ \mathbf{w}\perp\text{span}\{\mathbf{v}_1\}}} \mathbf{w}^T\mathbf{C}\mathbf{w}.$$

Following this pattern, the kth best (uncorrelated) way to combine the variables is $\mathbf{v}_k^T\mathbf{X}$, where $\mathbf{v}_k$ is the (orthonormal) eigenvector of $\mathbf{C}$ associated with the kth largest eigenvalue; $\mathbf{v}_k^T\mathbf{X}$ has variance λ_k.

Key concept. Let $\mathbf{X} = [X_1, \ldots, X_n]^T \in \mathbb{R}^n$ collect n random variables, with associated covariance matrix $\mathbf{C} \in \mathbb{R}^{n\times n}$. Since $\mathbf{C}$ is a symmetric positive-semidefinite matrix, it has real eigenvalues that we label

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

and corresponding *orthonormal* eigenvectors

$$\mathbf{v}_1,\ \mathbf{v}_2,\ \cdots,\ \mathbf{v}_n.$$

The *kth principal component* is the aggregated variable

$$Y_k := \mathbf{v}_k^T\mathbf{X}.$$

The variables $Y_1, \ldots, Y_k$ are *uncorrelated* and have variance

$$\text{Var}(Y_k) = \text{Var}(\mathbf{v}_k^T\mathbf{X}) = \mathbf{v}_k^T\mathbf{C}\mathbf{v}_k = \lambda_k.$$

We learn much about our variables from the relative size of the variances (eigenvalues)

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0.$$

If some of the latter eigenvalues are very small, that indicates that the set of n random variables can be well approximated by a fewer number of aggregated variables. These aggregated variables are the *principal components* of $X_1, \ldots, X_n$.

You might naturally ask, *How many principal components do I need to describe my data set?* Often one looks at a plot of the eigenvalues (for example, see Figure 6.11) and looks for a good cut-off. Since the aggregate variables are uncorrelated,

$$\text{Var}\Big(\sum_{j=1}^n Y_j\Big) = \sum_{j=1}^n \text{Var}(Y_j) = \sum_{j=1}^n \lambda_j.$$

We can thus regard the sum of the eigenvalues of $\mathbf{C}$ as a measure of the total variance in the variables. The proportion of this total

variance captured by the first k principal components is thus

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j}. \tag{6.4}$$

If you want, say, to capture 75% of the variation, then pick k large enough for the fraction (6.4) to exceed 0.75, and use the first k principal components. Figure 6.12 will show a plot of the fractions (6.4) for a sample data set.

6.3.3 Approximate PCA from empirical data

In practical situations, one often seeks to analyze empirical data drawn from some unknown distribution: the expected values and covariances are not available. Instead, we will *estimate* these from the measured data.

Suppose, as before, that we are considering n random variables, $X_1, \ldots, X_n$, with m samples of each:

$$x_{j,k}, \quad k = 1, \ldots, m,$$

i.e., $x_{j,k}$ is the kth sample of the random variable X_j. The expected value has the familiar *unbiased estimate*

$$\mu_j = \frac{1}{m} \sum_{k=1}^{m} x_{j,k}.$$

Similarly, we can approximate the covariance

$$\mathsf{Cov}(X_j, X_k) = \mathsf{E}[(X_j - \mathsf{E}[X_j])(X_k - \mathsf{E}[X_k)].$$

One might naturally estimate this as

$$\frac{1}{m} \sum_{\ell=1}^{m} (x_{j,\ell} - \mu_j)(x_{k,\ell} - \mu_k).$$

However, replacing the true expected values $\mathsf{E}[X_j]$ and $\mathsf{E}[X_k]$ with the empirical estimates μ_j and μ_k introduces some slight bias into this estimate. This bias can be removed by scaling, replacing $1/m$ by $1/(m-1)$ to get the *unbiased estimate*

$$s_{j,k} = \frac{1}{m-1} \sum_{\ell=1}^{m} (x_{j,\ell} - \mu_j)(x_{k,\ell} - \mu_k), \quad j, k = 1, \ldots, n.$$

If we let

$$\mathbf{x}_j = \begin{bmatrix} x_{j,1} \\ \vdots \\ x_{j,m} \end{bmatrix}, \quad j = 1, \ldots, n,$$

then each covariance estimate is just an inner product

$$s_{j,k} = \frac{1}{m-1}(\mathbf{x}_j - \mu_j)^T(\mathbf{x}_k - \mu_k).$$

Here the notation $\mathbf{x}_j - \mu_j$ means: subtract the scalar μ_j from all entries of the vector $\mathbf{x}_j$.

Thus, if we center the samples of each variable about its empirical mean, we can write the empirical covariance matrix $\mathbf{S} = [s_{j,k}]$ as a matrix product. Let

$$\mathfrak{X} := [\,(\mathbf{x}_1 - \mu_1) \quad (\mathbf{x}_2 - \mu_2) \quad \cdots \quad (\mathbf{x}_n - \mu_n)\,] \in \mathbb{R}^{m\times n},$$

so that

$$\mathbf{S} = \frac{1}{m-1}\mathfrak{X}^T\mathfrak{X}. \tag{6.5}$$

Now conduct principal component analysis just as before, but with the empirical covariance matrix $\mathbf{S}$ replacing the true covariance matrix $\mathbf{C}$. The eigenvectors of $\mathbf{S}$ now lead to *sample principal components*.

Where is the connection to the singular value decomposition? Notice how we formed the sample covariance matrix $\mathbf{S}$ in equation (6.5). Aside from the scaling $1/(m-1)$, this structure recalls the first step in our construction of the singular value decomposition earlier in the chapter. We can thus arrive at the sample principal components by computing the singular value decomposition of the data matrix $\mathfrak{X}$. This is why some say, "PCA is just the SVD." We summarize the details step-by-step.

To compute the singular value decomposition of some matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, start by computing the eigenvalues and eigenvectors of $\mathbf{A}^T\mathbf{A}$. In our setting, the eigenvectors of $\mathbf{S}$ are the right singular vectors of $\mathfrak{X}$.

1. Collect m samples of each of n random variables, $x_{j,k}$ for $j = 1,\ldots,n$ and $k = 1,\ldots,m$. (We need $m > 1$; typically $m \gg n$.)
2. Compute the empirical means of each column, $\mu_k = (\sum_{\ell=1}^{m} x_{k,\ell})/m$.
3. Stacking the samples of the kth variable in the vector $\mathbf{x}_k \in \mathbb{R}^m$, construct the mean-centered data matrix

$$\mathfrak{X} = [\,(\mathbf{x}_1 - \mu_1) \quad (\mathbf{x}_2 - \mu_2) \quad \cdots \quad (\mathbf{x}_n - \mu_n)\,] \in \mathbb{R}^{m\times n}.$$

4. Compute the (skinny) singular value decomposition $\mathfrak{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, with $\mathbf{U} \in \mathbb{R}^{m\times n}$, $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1,\ldots,\sigma_n) \in \mathbb{R}^{n\times n}$, and $\mathbf{V} = [\mathbf{v}_1 \ \cdots \ \mathbf{v}_n] \in \mathbb{R}^{n\times n}$.
5. The kth sample principal component is given by $\mathbf{v}_k^T\mathbf{X}$, where $\mathbf{X} = [X_1,\ldots,X_n]^T$ is the vector of random variables.
6. You can assess the importance of the various principal components via the eigenvalues of $\mathbf{S}$, given by $\lambda_k = \sigma_k^2/(m-1)$. If these eigenvalues decay rapidly as k increases, that is a sign that your data can be well-represented by the first few principal components.

A word of caution: when conducting principal component analysis, the *scale* of each column matters. For example, if the random variables sampled in each column of $\mathfrak{X}$ are measurements of physical quantities, they can differ considerably in magnitude depending on the units of measurement. By changing units of measurement, you can significantly alter the principal components.

6.3.4 *Clustering via PCA*

PCA can be used to cluster data. To illustrate, we turn to a data set comprising of chemical properties of Italian wines. The data set includes measurements of 13 different properties for 178 wines, giving a data matrix $\mathfrak{X}$ of dimension 178×13. (The properties include: alcohol content, malic acid, ash, alcalinity of the ash, etc.)

You can download the "Wine Data Set" from the UCI Machine Learning Repository, `https://archive.ics.uci.edu/ml/datasets/wine`. For more details on this data set and the application of eigenvector-based clustering to it, see: M. Forina, C. Armanino, M. Castino, and M. Ubigli. "Multivariate data analysis as a discriminating method of the origin of wines," *Vitis* 25 (1986) 189–201.

Each of these 178 wines comes from one of three grape varieties: Barolo (nebbiolo grape), Grignolino, or Barbera. Now a bottle of Barolo typically costs quite a bit more than the other two wines, so it would be interesting to know if these high-end wines really can be distinguished, chemically, from the others.

When working with a real data set, we must begin by preparing the data. Since variables may be measured in different units, we begin by computing the empirical mean of each variable, and dividing by the mean (so that each variable now has mean 1).

If you do not normalize your variables, you risk having an extremely large principal component dominated by one single variable that happens to have very large values.

With this normalization complete, conduct PCA as described above: form the data matrix $\mathfrak{X}$ and compute its dominant singular values and singular vectors. Figure 6.10 shows the singular values of $\mathfrak{X}$, suggesting that the first two or three principal components will dominate the others. Figure 6.11 shows the eigenvalues of $\mathbf{S} = \mathfrak{X}^T\mathfrak{X}/(m-1)$, which obey $\lambda_k = \sigma_k^2/(m-1)$. The squaring accentuates the differences between the large and small singular values. Illustrations like Figure 6.11, called *scree plots*, are commonly used by data scientists to decide how many principal components are worthy of consideration.

Using $\lambda_k = \sigma_k^2/(m-1)$, we have

$$\lambda_1 \approx 0.608, \qquad \lambda_2 \approx 0.289,$$
$$\lambda_3 \approx 0.161, \qquad \lambda_4 \approx 0.089.$$

Figure 6.12 helps us understand how much of the total variation in the data is contained in the first k principal components. It plots the ratio (6.4) using the eigenvalues of the empirical covariance matrix $\mathbf{S}$. By this measure, if you want to capture 75% of the variation in the data, you would use the first three principal components (since the data first exceeds 0.75 when $k = 3$).

How can we use principal components to *cluster* the data? Consider the kth sample of data, described by the variables

$$x_{1,k}, \quad x_{2,k}, \quad , \ldots, \quad , x_{13,k}.$$

Denote the first right singular vector of $\mathfrak{X} \in \mathbb{R}^{178\times 13}$ by

$$\mathbf{v}_1 = [\gamma_1, \ldots, \gamma_{13}]^T.$$

Then the variance-maximizing combination of the 13 variables is given by

$$\xi_k := \sum_{j=1}^{13} \gamma_j x_{j,k}.$$

Similarly, writing the second right singular vector of $\mathfrak{X}$ as

$$\mathbf{v}_2 = [\omega_1, \ldots, \omega_{13}]^T,$$

define the second-best variance maximizing combination as

$$\eta_k := \sum_{j=1}^{13} \omega_j x_{j,k}.$$

Here is the key idea: we have squeezed as much variance as possible from our 13 variables into 2 variables. Can we actually *reduce the dimension* of our data set from those 13 variables down into the two new variables?

$$(x_{1,k}, x_{2,k}, \ldots, x_{13,k}) \quad \Longrightarrow \quad (\xi_k, \eta_k)$$

If $\lambda_3 = \lambda_4 = \cdots = \lambda_{13} = 0$, then this would be a perfect compression of the variables. Of course in practice, the reduction is only approximate, but hopefully we have distilled the essential distinguishing features of the 13 variables into those 2 consolidated variables ξ_k and η_k. We can view (ξ_k, η_k) as a *projection* of the 13 dimensional data onto a two-dimensional space. Many other projections are possible (just take any two of the given variables), but the one from PCA is

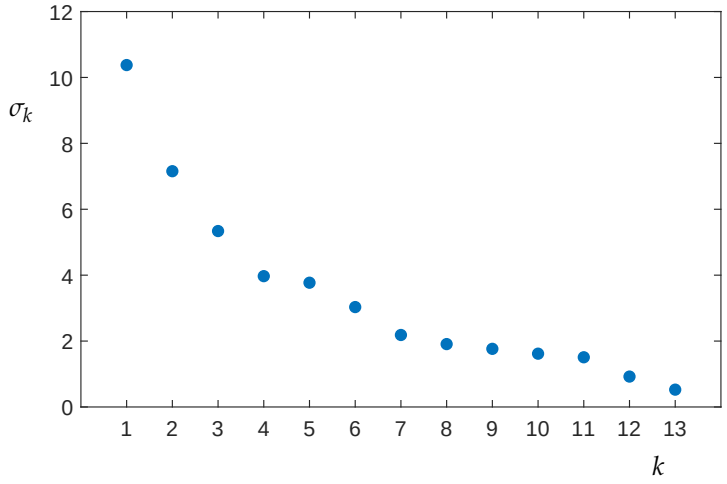


Figure 6.10: Singular values of the 178×13 wine data matrix $\mathfrak{X}$ (with normalized columns).

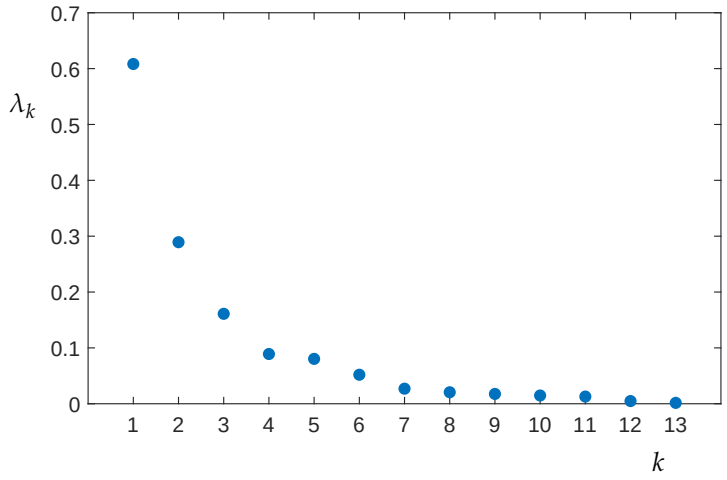


Figure 6.11: Eigenvalues of the 13×13 matrix $\mathbf{S} = (m-1)^{-1}\mathfrak{X}^T\mathfrak{X}$ for the wine data set. Such eigenvalue plots are called *scree plots*, as named by Raymond Cattell in 1966, who suggested a way of selecting the number of important principal components based on the elbow in the plot on the left. ("Scree" means the pile of rocks that often accumulate at the bottom of a cliff: your leading singular values form the cliff; the smaller singular values form the rubble on the ground.)

"This straight end portion we began calling the *scree*—from the straight line of rubble and boulders which forms at the pitch of sliding stability at the foot of a mountain. The initial implication was that this scree represents a "rubbish" of small error factors."
— Raymond Cattell, 1966

optimal (in the sense of maximizing variance). Figure 6.13 shows the (ξ_k, η_k) projection for these 178 data points.

Recall that our goal is to identify if each of these samples is Barolo, Grignolino, or Barbera. Can you see any clusters in Figure 6.13? To help, we apply the k-means algorithm with $k = 3$ to this data. Figure 6.14 shows the results.

We used MATLAB's `kmeans` implementation, running from 10 starting configurations and keeping the best clustering that results.

Conveniently enough, we have labeled data in this case, so we can check if the clustering in Figure 6.14 did a good job of identifying the three wine varieties. Figure 6.15 shows the results.

First of all, we notice that the three wine varieties really do look quite distinct, when projected into the two-dimensional (ξ_k, η_k) PCA coordinates. Even better, the k-means results match these pretty well: k-means made a few mistakes, especially at the frontier between Barolo and Grignolino, but overall it looks like we could do a de-

Indeed, k-means draws a cleaner boundary between these wines than we see in reality.

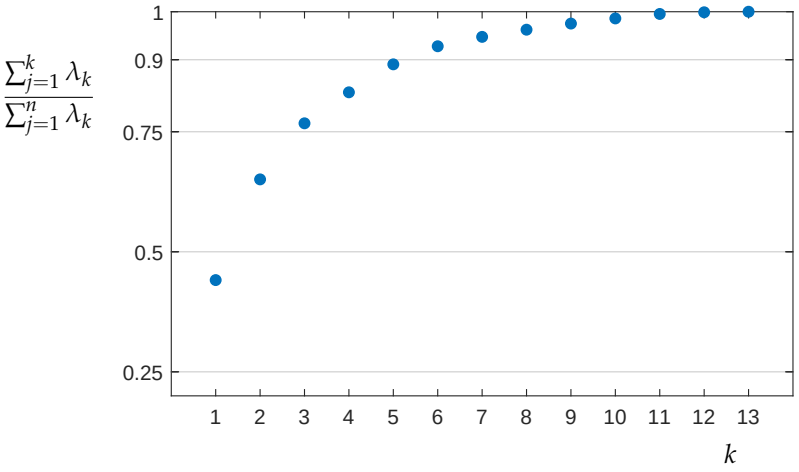


Figure 6.12: Cumulative sum of the eigenvalues of $\mathbf{S} = \mathfrak{X}^T\mathfrak{X}/(m-1)$, normalized by the total sum. This plot reveals *the percentage of the total variation is contained in the first k principal components*.

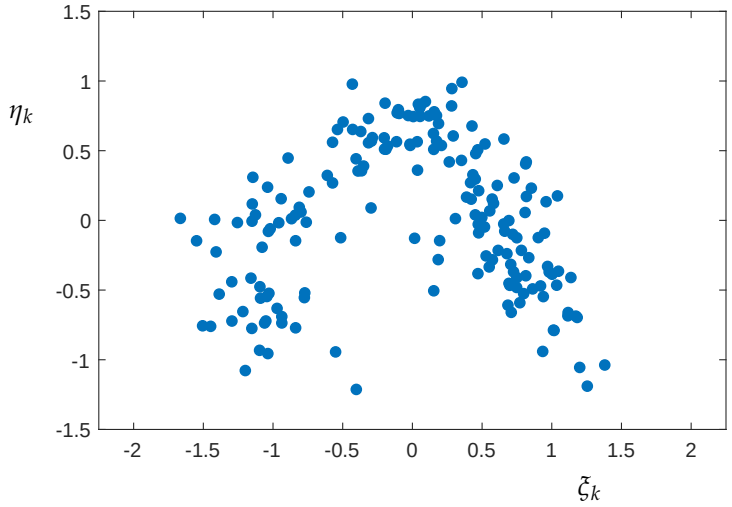


Figure 6.13: Projection of the wine data set into the two variables defined by the leading two principal components.

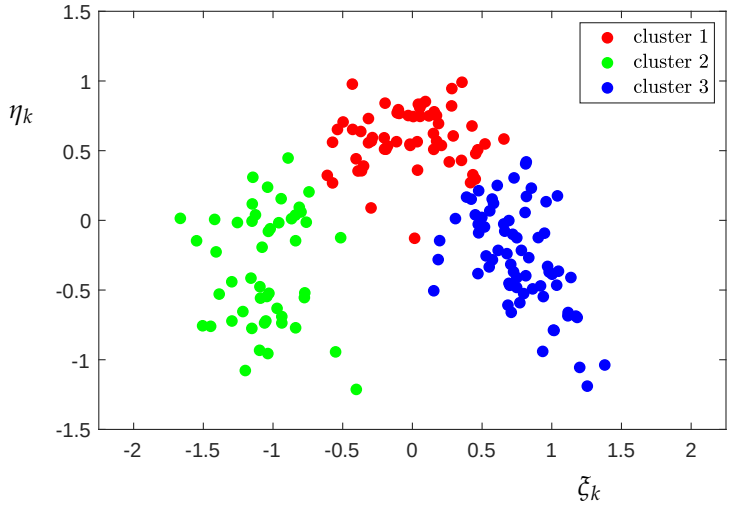


Figure 6.14: Results of k-means clustering of the results in Figure 6.13.

cent job of identifying wine through the combined efforts of PCA and k-means. (Whether, for this application, data science yields an improvement over the traditional manner of careful testing with a well-trained palate, I will let you be the judge. . . .)

The derivation above suggests that the PCA-derived variables (ξ_k, η_k) should exhibit greater variance than we would find from just picking a pair of the variables (say, $(x_{1,k}, x_{2,k})$. Figure 6.16 verifies this intuition, projecting the 13-dimensional data onto just two coordinates (we picked (1,2), (3,4), and (5,6)). (The color-coding refers to the true wine varieties, as in Figure 6.15.)

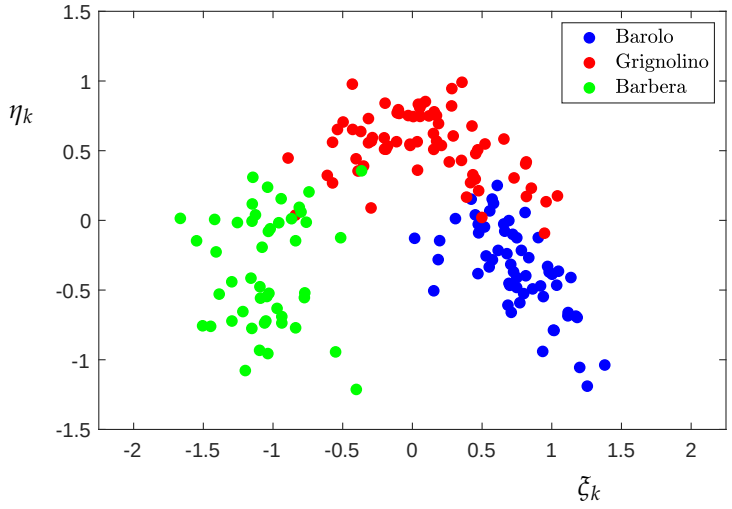


Figure 6.15: Repetition of Figure 6.13, but now color-coded according to the wine variety specified in the data set.

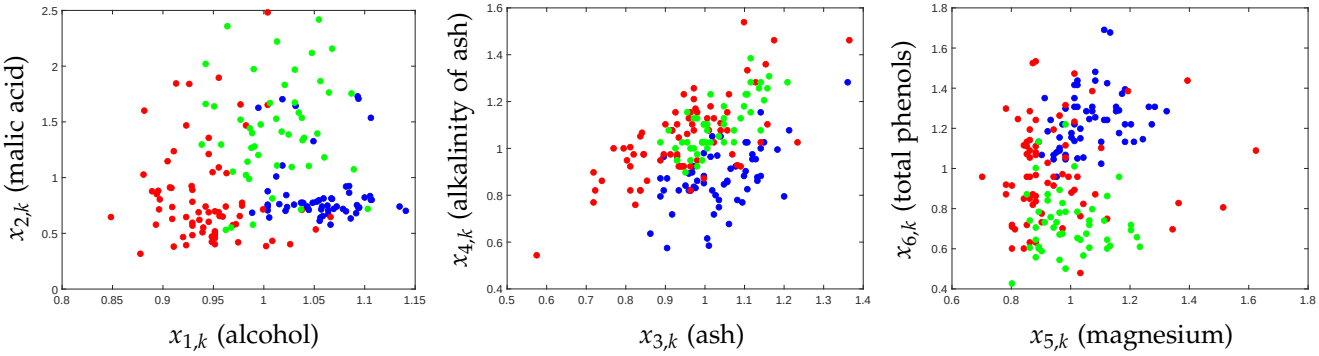


Figure 6.16: These three plots each project the 13-dimensional wine data set into two dimensions, just using the given variables instead of the derived variables from PCA. We see that these pairs variables, on their own, do a poor job of distinguishing the three classes of wines.

6.4 *Practical Computing: How Noise Affects the SVD*

The notion of *rank* is central to linear algebra, but in the early years of numerical computing, it became clear that a more nuanced understanding of rank would be necessary, to handle

We study linear algebra on a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ whose entries are *real numbers*. As there is an uncountably infinitude of real numbers, every entry $a_{j,k} \in \mathbb{R}$ can take on uncountably many values. When working on a computer, we cannot implement such an rich number system with infinitely many options. We settle instead for an excellent *floating point number system*, which has finitely many values spaced logarithmically from the very small to the very large. When we store a matrix on a computer, often $a_{j,k}$ is not one of these chosen values in the number system, and so it is rounded – by a relative amount around 10^{-16}, to the closest value in the number system.

The floating point number system is not "closed under addition and multiplication", meaning that quantities like $a_{j,k} + a_{r,s}$ and $a_{j,k}a_{r,s}$

need not be in the number system: so the results of such operations also need to be rounded. Through much careful and fruitful research, we now know how to implement a good number system, and work with algorithms in the presence of such small rounding errors.

The favored number system, IEEE floating-point arithmetic, is now almost universally used. The design of this system earned William Kahan the Turing Award (1989). Our understanding of the stability of algorithms in a floating-point environment was pioneered by James Hardy Wilkinson, also a Turing Award winner (1970).

The result is that we never really work with our matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, but instead with a nearby matrix

$$\widehat{\mathbf{A}} = \mathbf{A} + \mathbf{E}$$

for some small *perturbation* (you can think of it as *noise*) $\mathbf{E} \in \mathbb{R}^{m\times n}$. For good algorithms, we often have $\|\mathbf{E}\| \approx 10^{-16}\|\mathbf{A}\|$.

How does the perturbation $\mathbf{E}$ affect the rank?

6.5 *Randomized algorithms for approximation of the SVD*

6.6 *Recommender systems*

6.7 *Principal Orthogonal Decomposition (POD)*

6.8 *Interpolative approximations*

$$\mathbf{A} \approx \mathbf{CUR} = \mathbf{CX} = \mathbf{YR}.$$

6.9 *Afterword*

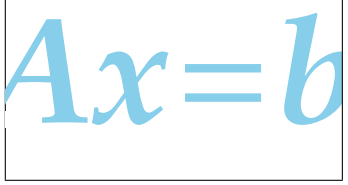
The singular value decomposition was developed in its initial form by Eugenio Beltrami (1873) and, independently, by Camille Jordan (1874).[2]

[2] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35:551–566, 1993

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 7
Linear Systems, the Pseudoinverse, and Ill-Posed Problems

Given mastery of the SVD and low-rank approximation, we are now prepared to develop a fully mature understanding of *systems of linear equations*. This understanding will develop in three steps of increasing sophistication:

- appreciate when the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ has an exact solution;
- use the *pseudoinverse* to solve this more general problem: of all solutions $\mathbf{x} \in \mathbb{R}^n$ to the least squares problem

  $$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$$

  find the one that minimizes $\|\mathbf{x}\|$;
- realize that for many important application problems, the pseudoinverse solution is prone to *instability*, and this can be remedied through a class of techniques called *regularization*.

This chapter will revisit some ideas introduced in Sections 5.3 and 5.9, as we focus deeply on $\mathbf{A}\mathbf{x} = \mathbf{b}$ problems.

7.1 Linear systems

Let us begin with four trivial examples that illustrate the breadth of the theory.

Example 7.1 (a unique solution exists)**.** Consider the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$,

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

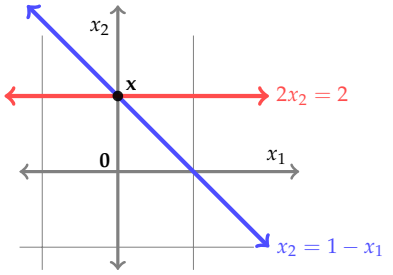
This system encodes two simple equations,

$$x_1 + x_2 = 1$$
$$2x_2 = 2.$$

The first equation gives $x_2 = 1 - x_1$, describing a line in the (x_1, x_2) plane (the blue line in figure on the right), while the second equation simply gives $2x_2 = 2$, another line in the (x_1, x_2) plane (the red line). These lines only intersect at a single point,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

the unique solution of $\mathbf{Ax} = \mathbf{b}$. ■



The blue line shows the first equation $x_1 + x_2 = 1$; the red line shows the second equation, $2x_2 = 2$; the solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$ is the unique point $\mathbf{x} = [0, 1]^T$ where the blue and red lines intersect.

Example 7.2 (a unique solution exists)**.** Suppose we add an additional equation to the last example,

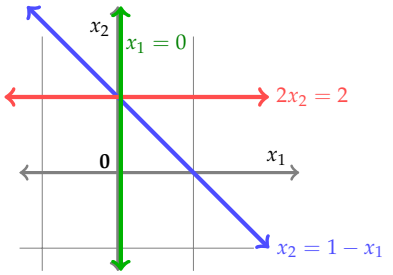
$$x_1 = 0,$$

which extends our $\mathbf{Ax} = \mathbf{b}$ problem to

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

Since $\mathbf{A}$ has more rows than columns, it cannot be invertible; however, *it is still possible, in some situations, for* $\mathbf{Ax} = \mathbf{b}$ *to have a unique solution.* The key here is that the new equation $x_1 = 0$ is *consistent* with the two preceding equations. The figure on the right shows this scenario, adding the green line $x_1 = 0$ that intersects the two preceding equations (in blue and red) at their common intersection: the three equations are consistent, simultaneously satisfied by

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$ ■



The equation $x_1 = 0$ adds the green line. Since the blue, red, and green lines all intersect at a common point ($\mathbf{x} = [x_1, x_2]^T = [0, 1]^T$), a unique solution exists to $\mathbf{Ax} = \mathbf{b}$.

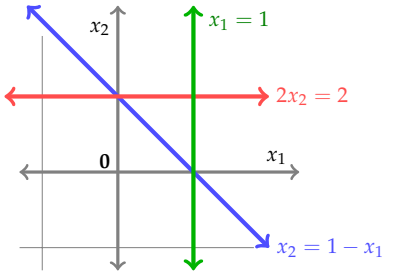
Example 7.3 (no solution exists)**.** Suppose we instead supplemented example 7.1 with the equation

$$x_1 = 1,$$

extending the $\mathbf{Ax} = \mathbf{b}$ problem to

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

Now the new equation (again shown in green in the plot at the right)



The equation $x_1 = 1$ adds the green line. For $\mathbf{Ax} = \mathbf{b}$ to have a solution, we need all three of these lines to intersect at a single point. Since that never happens, $\mathbf{Ax} = \mathbf{b}$ never has a solution.

does not pass through the intersection of the two original equations (blue and red lines). For the system $\mathbf{Ax} = \mathbf{b}$ to have a solution, all three of the constituent scalar equations must be *consistent*, meaning they must intersect in a single point in the (x_1, x_2) plane. That is never the case here, *so* $\mathbf{Ax} = \mathbf{b}$ *has no solution*. The system is *overdetermined*.

The situation is not hopeless: we can instead solve the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{Ax}\|, \tag{7.1}$$

which always has a solution. Using techniques explored in Section 5.9 (see especially equation (5.18)) and unpacked in the next section, since the columns of $\mathbf{A}$ are linearly independent we can solve the least squares problem (7.1) by solving

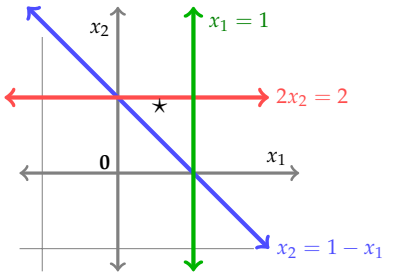
$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b},$$

i.e.,

$$\begin{bmatrix} 2 & 1 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix},$$

which has the unique solution (black star on the plot to the right) that we will denote

$$\mathbf{x}_+ = \begin{bmatrix} 5/9 \\ 8/9 \end{bmatrix}. \quad \blacksquare$$



The black star shows the solution $\mathbf{x}_+ = [5/9, 8/9]^T$ to the least squares problem (7.1); this point is not *on* any of the three lines (so none of the three equations $x_1 + x_2 = 1$, $2x_2 = 2$, $x_1 = 1$ is satisfied exactly), but it is *near* these lines.

Example 7.4 (infinitely many solutions; unique minimum norm solution)**.** Now adapt Example 7.1 by changing the $(2,2)$ entry of $\mathbf{A}$:

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

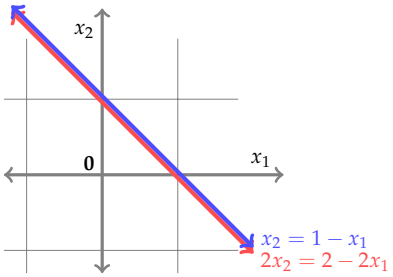
Now both scalar equations are equivalent,

$$\begin{aligned} x_1 + x_2 &= 1 \\ 2x_1 + 2x_2 &= 2. \end{aligned}$$

In the plot on the right, we see the two equivalent conditions tracing out the same line, $x_2 = 1 - x_1$. *Any* $\mathbf{x}$ *on this line satisfies both scalar equations, and hence solves* $\mathbf{Ax} = \mathbf{b}$*:*

$$\mathbf{x} = \begin{bmatrix} x_1 \\ 1 - x_1 \end{bmatrix}.$$

The equation $\mathbf{Ax} = \mathbf{b}$ *has infinitely many solutions, and we say the system is* underdetermined.



The blue line shows the first equation $x_1 + x_2 = 1$; the red line shows the second equation, $2x_2 = 2$; the solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$ is the unique point $\mathbf{x} = [0, 1]^T$ where the blue and red lines intersect.

In this case, we have the opportunity to impose an additional condition on the solution so as to get a unique solution. In many

applications it is appealing to minimize $\|\mathbf{x}\|$, which is equivalent to minimizing

$$\|\mathbf{x}\|^2 = x_1^2 + x_2^2.$$

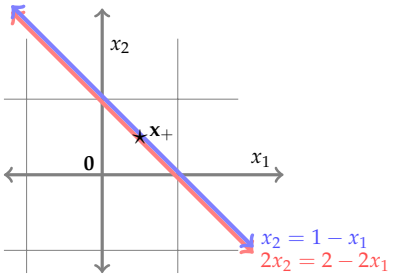
For the specific solution to this problem, write out

$$\|\mathbf{x}\|^2 = (x_1)^2 + (1 - x_1)^2 = 2x_1^2 - 2x_1 + 1,$$

take a derivative with respect to x_1 and set to zero to find

$$\mathbf{x}_+ = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}.$$

The plot on the right shows this point as a star; notice that it is simply the point on the line that is closest to the origin. ■



The star shows the minimum norm solution to $\mathbf{Ax} = \mathbf{b}$, given by $\mathbf{x}_+ = [1/2, 1/2]^T$.

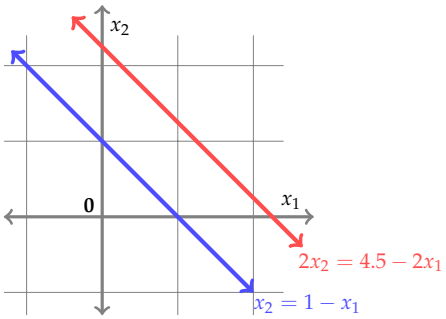
Example 7.5 (no solutions; infinitely many least squares solutions)**.** Finally, consider a tweak to the last example, changing the second entry of $\mathbf{b}$:

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4.5 \end{bmatrix}.$$

Now the two scalar equations are *contradictory*:

$$x_1 + x_2 = 1$$
$$2x_1 + 2x_2 = 4.5.$$

These scalar equations describe lines that never intersect, and so there is no $\mathbf{x} \in \mathbb{R}^2$ that satisfies both equations.



The two scalar equations describe lines that never intersect: $\mathbf{Ax} = \mathbf{b}$ has no solution.

No worries: when we encountered this situation in Example 7.3 we sought the *least squares solution* by solving the *normal equations*

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b},$$

which in this case amounts to

$$\begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \end{bmatrix}.$$

But in this case a strange situation arises. The matrix $\mathbf{A}^T\mathbf{A}$ is not invertible, so the normal equations – and hence the least squares problem – has *infinitely many solutions*: any $\mathbf{x} = [x_1, x_2]^T$ that satisfies

$$5x_1 + 5x_2 = 10$$

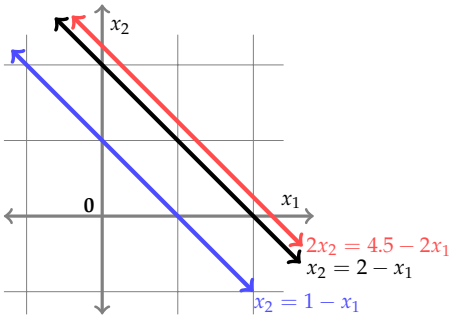
will be least squares solution. Thus any vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ 2 - x_1 \end{bmatrix}$$

minimizes $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$. The plot on the right shows these solutions as the black arrow. To verify that any such $\mathbf{x}$ really does solve the least squares problem, compute the residual

$$\mathbf{b} - \mathbf{A}\mathbf{x} = \begin{bmatrix} 1 \\ 4.5 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ 2 - x_1 \end{bmatrix}$$
$$= \begin{bmatrix} 1 - x_1 - (2 - x_1) \\ 4.5 - 2x_1 - 2(2 - x_1) \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix},$$

which is indeed independent of x_1.



The thick black arrow denotes the infinitely many solutions of the least squares problem $\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$. Any $\mathbf{x}$ on this black line gives the same minimal value of $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$.

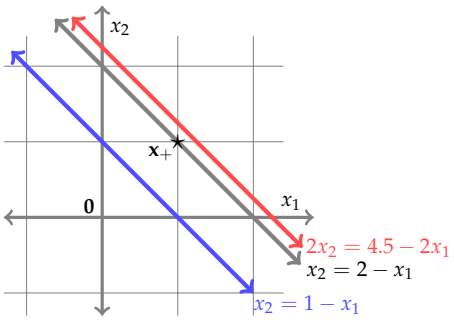
Following the lead of Example 7.4, we can still get a minimum norm solution by minimizing $\|\mathbf{x}\|$ over all solutions to the least squares problem. For such solutions $\mathbf{x}$,

$$\|\mathbf{x}\|^2 = (x_1)^2 + (2 - x_1)^2 = 2x_1^2 - 4x_1 + 4,$$

which takes a minimum when $x_1 = 1$, i.e,

$$\mathbf{x}_+ = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

which is shown as a star on the plot on the right. ■



The black star shows the $\mathbf{x}$ with minimum norm among the infinitely many solutions of the least squares problem $\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$.

This last example shows the most general scenario we can encounter when attempting to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$. In the next section, we will dive deeper into the mechanics illustrated by all the examples in this section, developing a way to use the SVD to write down the unique vector $\mathbf{x}_+$ that has minimum norm among all the vectors $\mathbf{x}$ that minimize $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$. Before developing this theory, let us consider the five examples in this section using the language of elementary linear algebra. We use two principles.

- A solution $\mathbf{x}$ to $\mathbf{A}\mathbf{x} = \mathbf{b}$ *exists* if and only if $\mathbf{b} \in \mathcal{R}(\mathbf{A})$.
- A solution $\mathbf{x}$ to $\mathbf{A}\mathbf{x} = \mathbf{b}$ is *unique* if and only if $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$.

When reading a flowchart, the diamond-shaped boxes denote a decision point: depending on the answer, the flow proceeds in one direction or the other.

The flowchart in Figure 7.1 delineates the possibilities that follow from these two principles.

The first statement follows immediately from the definition of the column space (range):

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$$

means that $\mathcal{R}(\mathbf{A})$ is simply the collection of all vectors $\mathbf{b}$ for which $\mathbf{A}\mathbf{x} = \mathbf{b}$ has a solution.

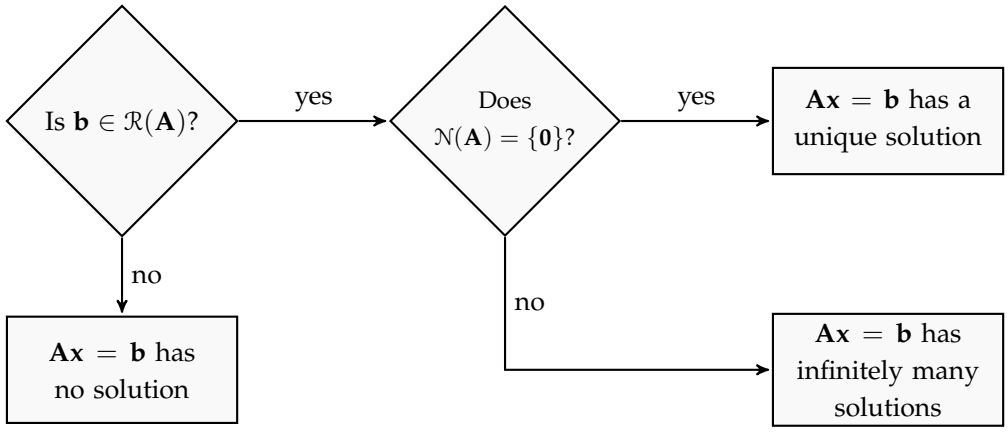


Figure 7.1: Decision process for analyzing if $\mathbf{Ax} = \mathbf{b}$ has a solution $\mathbf{x}$ for a given choice of $\mathbf{A}$ and $\mathbf{b}$.

The second statement follows from the fact that if $\mathbf{z} \in \mathcal{N}(\mathbf{A})$, then $\mathbf{Az} = \mathbf{0}$, so for any solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$,

$$\mathbf{A}(\mathbf{x} + \mathbf{z}) = \mathbf{Ax} + \mathbf{Az} = \mathbf{b} + \mathbf{0} = \mathbf{b},$$

and hence $\mathbf{x} + \mathbf{z}$ is also a solution.

If $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$, then we must have $\mathbf{z} = \mathbf{0}$, and so $\mathbf{x} + \mathbf{z} = \mathbf{x}$, and this procedure does not create a new solution.

Let us interpret the examples in this section in light of these principles.

- Example 7.1: $\mathcal{R}(\mathbf{A}) = \mathbb{R}^2$ and $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$, so $\mathbf{Ax} = \mathbf{b}$ has a unique solution for all $\mathbf{b} \in \mathbb{R}^2$.
- Example 7.2: $\mathcal{R}(\mathbf{A})$ is a two-dimensional subspace of $\mathbb{R}^3$ with $\mathbf{b} = [1, 2, 0]^T \in \mathcal{R}(\mathbf{A})$ and $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$, so $\mathbf{Ax} = \mathbf{b}$ has a unique solution *for this particular* $\mathbf{b}$.
- Example 7.3 uses the same $\mathbf{A}$ as the last example, but now $\mathbf{b} = [1, 2, 1]^T \notin \mathcal{R}(\mathbf{A})$, and so $\mathbf{Ax} = \mathbf{b}$ does not have a solution.
- Example 7.4: $\mathcal{R}(\mathbf{A})$ and $\mathcal{N}(\mathbf{A})$ are both one-dimensional subspaces of $\mathbb{R}^2$. For $\mathbf{b} = [1, 2]^T \in \mathcal{R}(\mathbf{A})$ a solution exists, but it is not unique since any vector $\mathbf{z} = \zeta[1, -1]^T \in \mathcal{N}(\mathbf{A})$.
- Example 7.5 uses the same $\mathbf{A}$ as the last example, but now $\mathbf{b} = [1, 4.5]^T \notin \mathcal{R}(\mathbf{A})$, so $\mathbf{Ax} = \mathbf{b}$ does not have a solution.

7.2 *The pseudoinverse*

We now have a full understanding of the $\mathbf{Ax} = \mathbf{b}$ problem; we now seek a similarly complete understanding of the general least squares problem.

Least Squares problem. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Among all $\mathbf{x} \in \mathbb{R}^n$ that solve

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|,$$

find the $\mathbf{x}$ that minimizes $\|\mathbf{x}\|$.

In some applications, minimizing $\|\mathbf{x}\|$ could amount to *minimizing energy*. Some settings would prefer to pluck out a different $\mathbf{x}$ from all solutions of the least squares problem. A popular alternative is to *minimize the number of nonzeros in* $\mathbf{x}$, associated with *sparse approximation* (*compressive sensing*); since this goal is difficult to implement, one instead minimizes the "one-norm"

$$\|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n|.$$

This goal relates to *LASSO regression*, as we will mention later in this chapter.

We will now develop a full solution to this problem, starting by revisiting the steps we traced out in Section 5.9 for the case where $\text{rank}(\mathbf{A}) = n$. Here we merely assume that $\text{rank}(\mathbf{A}) = r$, which could be less than n and m.

As in Section 5.9, start by decomposing $\mathbf{b}$ into the form

$$\mathbf{b} = \mathbf{b}_R + \mathbf{b}_N$$

with $\mathbf{b}_R \in \mathcal{R}(\mathbf{A})$ and $\mathbf{b}_N \in \mathcal{N}(\mathbf{A}^T)$. Now we seek to minimize $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$, which is equivalent to minimizing

This decomposition follows from the FTLA (Theorem 5.5), since $\mathbb{R}^m = \mathcal{R}(\mathbf{A}) \oplus \mathcal{N}(\mathbf{A}^T)$. Recall also that since $\mathcal{R}(\mathbf{A}) \perp \mathcal{N}(\mathbf{A}^T)$, the elements in this decomposition of $\mathbf{b}$ are orthogonal: $\mathbf{b}_R \perp \mathbf{b}_N$.

$$\begin{aligned}\|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 &= \|(\mathbf{b}_R + \mathbf{b}_N) - \mathbf{A}\mathbf{x}\|^2 \\ &= \|(\mathbf{b}_R - \mathbf{A}\mathbf{x}) + \mathbf{b}_N\|^2.\end{aligned}$$

Since $\mathbf{b}_R \in \mathcal{R}(\mathbf{A})$ and $\mathbf{A}\mathbf{x} \in \mathcal{R}(\mathbf{A})$, and $\mathcal{R}(\mathbf{A})$ is a *subspace*, we must have $\mathbf{b}_R - \mathbf{A}\mathbf{x} \in \mathcal{R}(\mathbf{A})$.

Note that $\mathbf{b}_R - \mathbf{A}\mathbf{x} \in \mathcal{R}(\mathbf{A})$ and anything in $\mathcal{R}(\mathbf{A})$ is orthogonal to $\mathbf{b}_N \in \mathcal{N}(\mathbf{A}^T)$, so we can invoke the Pythagorean Theorem (Theorem 2.1) to obtain

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 = \|\mathbf{b}_R - \mathbf{A}\mathbf{x}\|^2 + \|\mathbf{b}_N\|^2.$$

The choice of $\mathbf{x}$ cannot affect $\|\mathbf{b}_N\|^2$, which we can regard as the *unreachable* part of the error. On the other hand, since $\mathbf{b}_R \in \mathcal{R}(\mathbf{A})$, we can always find some $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}_R$. Here is the trick: we need to find this $\mathbf{x}$ while only having $\mathbf{b}$ without specifically knowing $\mathbf{b}_R$. Thankfully a simple trick resolves the problem. If $\mathbf{A}\mathbf{x} = \mathbf{b}_R$, then

Recall that $\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$, so if $\mathbf{b}_R \in \mathcal{R}(\mathbf{A})$, it must equal $\mathbf{A}\mathbf{x}$ for some $\mathbf{x}$.

$$\mathbf{b} - \mathbf{A}\mathbf{x} = (\mathbf{b}_R - \mathbf{A}\mathbf{x}) + \mathbf{b}_N = \mathbf{b}_N.$$

Since $\mathbf{b}_N \in \mathcal{N}(\mathbf{A}^T)$, premultiply this equation by $\mathbf{A}^T$ to get

$$\mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{A}^T\mathbf{b}_N = \mathbf{0}.$$

Rearrange to get

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}, \tag{7.2}$$

which, as we saw in Section 5.9, are called the *normal equations*.

In Section 5.9, we found $\mathbf{x}$ simply by inverting $\mathbf{A}^T\mathbf{A}$:

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

However, if $\text{rank}(\mathbf{A}) = r < n$, then $\mathbf{A}^T\mathbf{A}$ will not be invertible. We encountered this situation in Example 7.5, where

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

is clearly not invertible. No surprise, the SVD will illuminate the path forward. Recall the reduced, full, and dyadic forms:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} \mathbf{U} & \mathbf{U}_\perp \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}^T \\ \mathbf{V}_\perp^T \end{bmatrix} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

Since

$$\mathbb{R}^n = \mathcal{R}(\mathbf{A}^T) \oplus \mathcal{N}(\mathbf{A}) = \mathcal{R}(\mathbf{V}) \oplus \mathcal{R}(\mathbf{V}_\perp),$$

This decomposition is another aspect of the FTLA (Theorem 5.5); recall further that $\mathcal{R}(\mathbf{A}^T) \perp \mathcal{N}(\mathbf{A})$.

any vector $\mathbf{x} \in \mathbb{R}^n$ can be written as

$$\mathbf{x} = \mathbf{x}_R + \mathbf{x}_N = \mathbf{V}\mathbf{c} + \mathbf{V}_\perp \mathbf{d} = \sum_{j=1}^{r} c_j \mathbf{v}_j + \sum_{j=r+1}^{n} d_{j-r} \mathbf{v}_j,$$

where $\mathbf{x}_R = \mathbf{V}\mathbf{c} \in \mathcal{R}(\mathbf{A}^T)$ and $\mathbf{x}_N = \mathbf{V}_\perp \mathbf{d} \in \mathcal{N}(\mathbf{A})$, with $\mathbf{c} \in \mathbb{R}^r$ and $\mathbf{d} \in \mathbb{R}^{n-r}$. Find formulas for $\mathbf{c}$ and $\mathbf{d}$, and we will have solved for $\mathbf{x}$.

The elements of $\mathbf{c}$ and $\mathbf{d}$ collect the coefficients of $\mathbf{x}_R$ and $\mathbf{x}_N$ expanded in the bases $\{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$ and $\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}$.

We know that $\mathbf{x}$ must satisfy the normal equations (7.2). Into this formula

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

substitute the reduced SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ and the decomposition $\mathbf{x} = \mathbf{V}\mathbf{c} + \mathbf{V}_\perp\mathbf{d}$ to get

$$(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)\big(\mathbf{V}\mathbf{c} + \mathbf{V}_\perp\mathbf{d}\big) = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T\mathbf{b}.$$

Now distribute the transpose to get

$(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T = (\mathbf{V}^T)^T\mathbf{\Sigma}^T\mathbf{U}^T = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T$, since $\mathbf{\Sigma}$ is a square diagonal matrix.

$$\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\big(\mathbf{V}\mathbf{c} + \mathbf{V}_\perp\mathbf{d}\big) = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b}.$$

We can simplify this expression into beautiful form. Since the columns of $\mathbf{U}$ (the *left* singular vectors) are orthonormal, $\mathbf{U}^T\mathbf{U} = \mathbf{I}$,

$$\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T\big(\mathbf{V}\mathbf{c} + \mathbf{V}_\perp\mathbf{d}\big) = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b}.$$

Distribute across the expression for $\mathbf{x}$ to get

$$\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T\mathbf{V}\mathbf{c} + \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T\mathbf{V}_\perp\mathbf{d} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b}.$$

Now the orthonormality of the *right* singular vectors ensures that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V}_\perp = \mathbf{0}$. These facts simplify the expression considerably,

$$\mathbf{V}\mathbf{\Sigma}^2\mathbf{c} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b},$$

wiping out $\mathbf{d}$. Premultiply both sides first by $\mathbf{V}^T$ to get

$$\mathbf{\Sigma}^2\mathbf{c} = \mathbf{\Sigma}\mathbf{U}^T\mathbf{b}$$

and then by $(\mathbf{\Sigma}^2)^{-1} = (\mathbf{\Sigma}^{-1})^2$ to arrive, finally, at a clean expression for $\mathbf{c} \in \mathbb{R}^r$:

$$\mathbf{c} = \mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}. \tag{7.3}$$

Recall that $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{r\times r}$ is invertible, since the rank r equals the number of nonzero singular values:

$$\mathbf{\Sigma}^{-1} = \operatorname{diag}(1/\sigma_1, \ldots, 1/\sigma_r).$$

What about $\mathbf{d}$, the coefficients that reveal the piece $\mathbf{x}_N = \mathbf{V}_\perp \mathbf{d}$ in $\mathcal{N}(\mathbf{A})$, that washed out of our calculation above? *The vector* $\mathbf{d} \in \mathbb{R}^{n-r}$ *can be anything!* Regardless of the choice of $\mathbf{d}$,

$$\begin{aligned} \mathbf{x} &= \mathbf{V}\mathbf{c} + \mathbf{V}_\perp \mathbf{d} \\ &= \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} + \mathbf{V}_\perp \mathbf{d} \end{aligned} \tag{7.4}$$

will solve the normal equations $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$, and hence the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|.$$

Indeed, equation (7.4) describes the general form of all solutions to the least squares problem.

We are hunting for the minimum norm solution of the least squares problem. Since $\mathbf{x}_R = \mathbf{V}\mathbf{c}$ and $\mathbf{x}_N = \mathbf{V}_\perp \mathbf{d}$ are orthogonal, we have

$$\|\mathbf{x}\|^2 = \|\mathbf{x}_R + \mathbf{x}_N\|^2 = \|\mathbf{x}_R\|^2 + \|\mathbf{x}_N\|^2,$$

Here we use the Pythagorean Theorem (Theorem 2.1) again.

where $\mathbf{x}_N = \mathbf{V}_\perp \mathbf{d}$ has

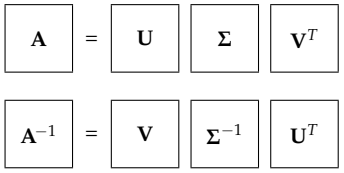
$$\|\mathbf{x}_N\|^2 = (\mathbf{V}_\perp \mathbf{d})^T(\mathbf{V}_\perp \mathbf{d}) = \mathbf{d}^T\mathbf{V}_\perp^T\mathbf{V}_\perp \mathbf{d} = \mathbf{d}^T\mathbf{d} = \|\mathbf{d}\|^2.$$

Hence, to make $\|\mathbf{x}\|$ as small as possible,
we must choose $\mathbf{d} = \mathbf{0}$, so that $\mathbf{x}_N = \mathbf{V}_\perp \mathbf{d} = \mathbf{0}$.

We denote the minimum norm solution of the least squares problem

$$\mathbf{x}_+ = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \sum_{j=1}^{r} \frac{1}{\sigma_j}(\mathbf{u}_j^T\mathbf{b})\mathbf{v}_j. \tag{7.5}$$

Invertible case, $r = m = n$:



Notice the matrix $\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T$ in this expression. If $\mathbf{A}$ is a square invertible matrix ($r = m = n$), then each of the $\mathbf{U}$, $\boldsymbol{\Sigma}$, and $\mathbf{V}$ matrices in the SVD are square and invertible, and we have

$$\begin{aligned} \mathbf{A}^{-1} &= (\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^{-1} \\ &= (\mathbf{V}^T)^{-1}\boldsymbol{\Sigma}^{-1}\mathbf{U}^{-1} \\ &= \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T \end{aligned}$$

Note that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ imply that $(\mathbf{V}^T)^{-1} = \mathbf{V}$ and $\mathbf{U}^{-1} = \mathbf{U}^T$.

Thus, in this special case of invertible $\mathbf{A}$, equation (7.5) reduces to the comforting form

$$\mathbf{x}_+ = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \mathbf{A}^{-1}\mathbf{b}.$$

Even for more general $\mathbf{A}$ (m, n, and r are not all the same)

the matrix $\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T$ *acts like an inverse*.

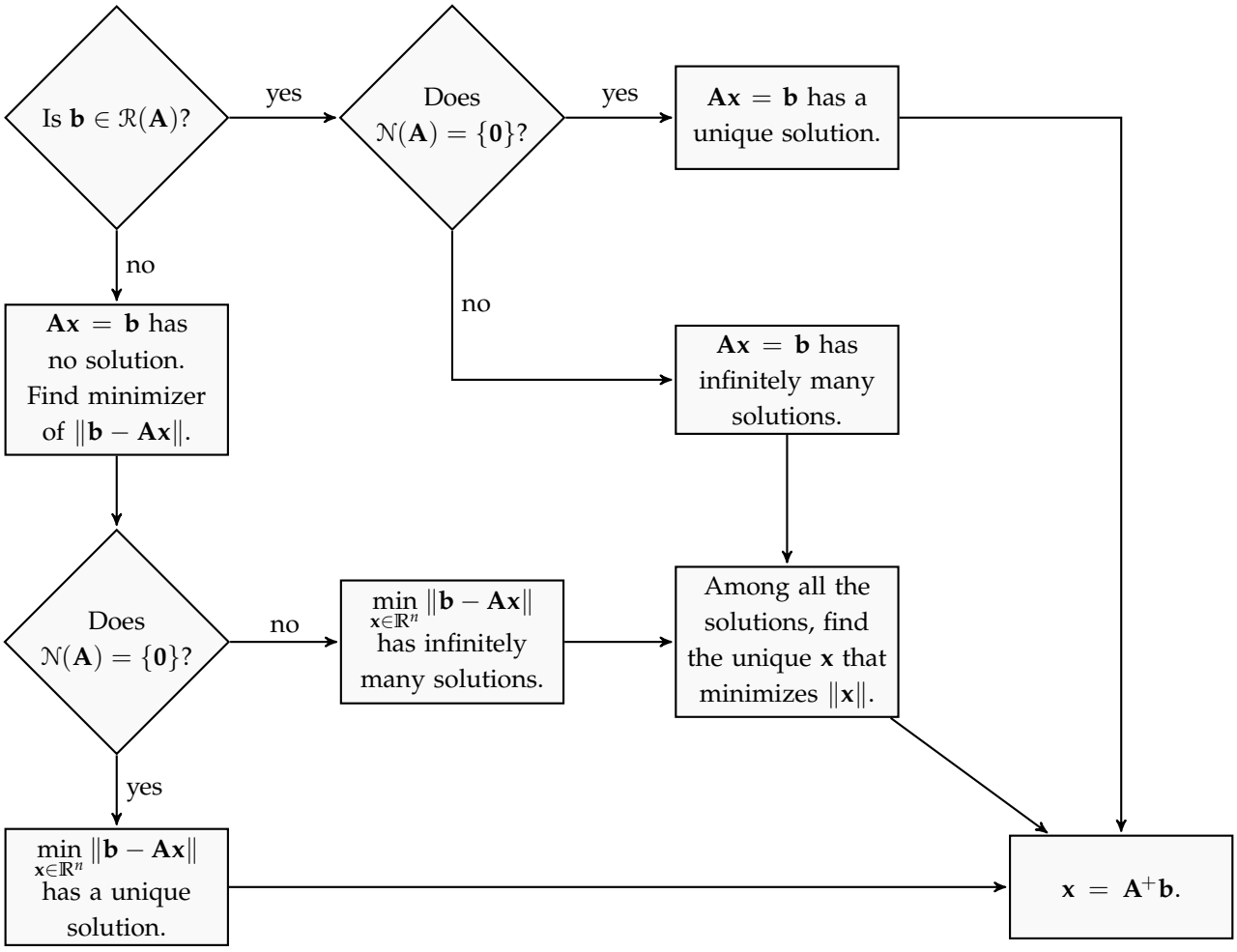


Figure 7.2: Revisiting the $\mathbf{Ax} = \mathbf{b}$ flowchart from Figure 7.1, but now adding in the least squares problem. When infinite solutions exist, we pick the unique one of minimal norm. We shall see in the pages ahead that the unique smallest norm solution can always be written as $\mathbf{x} = \mathbf{A}^+\mathbf{x}$.

This motivates us to call this matrix the *pseudoinverse* of $\mathbf{A}$, denoted

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T.$$

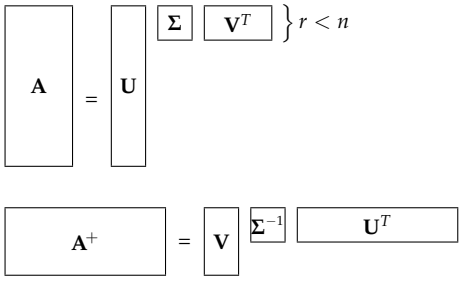
Just as it is sometimes helpful to write the SVD in dyadic form, we do the same for the pseudoinverse:

$$\mathbf{A}^+ = \sum_{j=1}^{n} \frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j^T$$

The dyadic formulas for $\mathbf{A}$ and $\mathbf{A}^+$ look very similar: $\mathbf{A}^+$ replaces σ_j with $1/\sigma_j$ and swaps the roles of $\mathbf{u}_j$ and $\mathbf{v}_j$. Notice that if $\mathbf{A} \in \mathbb{R}^{m\times n}$, then $\mathbf{A}^+ \in \mathbb{R}^{n\times m}$, so the pseudoinverse has the same *shape* as $\mathbf{A}^T$. (Don't confuse $\mathbf{A}^+$ with $\mathbf{A}^T$, they will generally be very different objects!) See the sketch in the margin as an example.

Figure 7.2 expands the earlier flowchart for $\mathbf{Ax} = \mathbf{b}$ problems in Figure 7.1 to find the minimal norm solution of the general least

General case (showing $r < n < m$):



squares problem. The solution to all these problems can always be written simply as

$$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b}.$$

Definition 7.1. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be a rank-r matrix with (reduced and dyadic) SVD*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

The pseudoinverse *of* $\mathbf{A}$*, denoted* $\mathbf{A}^+ \in \mathbb{R}^{n\times m}$ *is*

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \sum_{j=1}^{r} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T.$$

Despite its importance in mathematics and statistics, the notation for the pseudoinverse has not entirely setted down; for example, you will sometimes see $\mathbf{A}^+$ instead written as $\mathbf{A}^\dagger$ (read "$\mathbf{A}$-*dagger*").

Let us shine a light on a few special examples.

- When $\mathbf{A} = \mathbf{0} \in \mathbb{R}^{m\times n}$, we have $r = 0$. How should we interpret the formulas in Definition 7.1? We simply define

  $$\mathbf{0}^+ = \mathbf{0}.$$

  This formula might seem strange, but it makes sense if we insist that $\mathbf{x}_+ = \mathbf{A}^+\mathbf{b}$ be the minimum norm solution to the least squares problem: if $\mathbf{A} = \mathbf{0}$, then $\|\mathbf{b} - \mathbf{A}\mathbf{x}\| = \|\mathbf{b}\|$: any $\mathbf{x} \in \mathbb{R}^n$ solves the least squares problem, and $\mathbf{x}_+ = \mathbf{0} = \mathbf{0}\mathbf{b}$ is the solution with minimal norm.

- If $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a square, invertible matrix, then

  $$\mathbf{A}^+ = \mathbf{A}^{-1}.$$

  In this special case, $\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+\mathbf{A} = \mathbf{I} \in \mathbb{R}^{n\times n}$: the pseudoinverse is the inverse.

- If $\mathbf{A} \in \mathbb{R}^{m\times n}$ has linearly independent *columns* (so $r = n \leq m$), then $\mathbf{A}^T\mathbf{A}$ is invertible (can you explain why?) and hence

  $$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T,$$

  agreeing with the formula for the pseudoinverse given in Section 7.2. In this special case $\mathbf{A}^+\mathbf{A} = \mathbf{I} \in \mathbb{R}^{n\times n}$ but, if $n < m$, then $\mathbf{A}\mathbf{A}^+ \neq \mathbf{I} \in \mathbb{R}^{m\times m}$: the pseudoinverse is a "left inverse" of $\mathbf{A}$ but not a "right inverse".

- If $\mathbf{A} \in \mathbb{R}^{m\times n}$ has linearly independent *rows* (so $r = m \leq n$), then $\mathbf{A}\mathbf{A}^T$ is invertible (why?) and hence

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

Can you derive this expression? Hint: Substitute the reduced SVD on the right and simplify to get the formula for $\mathbf{A}^+$ in Definition 7.1.

In this special case $\mathbf{A}\mathbf{A}^+ = \mathbf{I} \in \mathbb{R}^{m\times m}$ but, if $m < n$, then $\mathbf{A}^+\mathbf{A} \neq \mathbf{I} \in \mathbb{R}^{n\times n}$: the pseudoinverse is a "right inverse" but not a "left inverse" of $\mathbf{A}$.

Now we compute a few small pseudoinverse for the examples at the start of this chapter.

Example 7.6 (Pseudoinverse for Examples 7.2 and 7.3)**.** Examples 7.2 and 7.3 used

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

Since $\text{rank}(\mathbf{A}) = r = n = 2$, we can use the simple formula

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$$

to arrive at

$$\mathbf{A}^+ = \begin{bmatrix} 4/9 & -2/9 & 5/9 \\ 1/9 & 4/9 & -1/9 \end{bmatrix}. \qquad \blacksquare$$

Example 7.7 (Pseudoinverse for Examples 7.4 and 7.5)**.** These examples used

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$$

which has rank $r = 1$ and reduced SVD

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} \begin{bmatrix} \sqrt{10} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}.$$

The pseudoinverse is thus

$$\begin{aligned} \mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T &= \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{10} \end{bmatrix} \begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix} \\ &= \begin{bmatrix} 1/10 & 2/10 \\ 1/10 & 2/10 \end{bmatrix}. \end{aligned}$$

Let us emphasize the *pseudo* part of *pseudoinverse*. Notice that

$$\mathbf{A}^+\mathbf{A} = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

while

$$\mathbf{A}\mathbf{A}^+ = \begin{bmatrix} 1/5 & 2/5 \\ 2/5 & 4/5 \end{bmatrix},$$

so neither $\mathbf{A}^+\mathbf{A}$ nor $\mathbf{A}\mathbf{A}^+$ agree with the identity: a consequence of the fact that $\mathbf{A}$ as not full-rank, $1 = r < n = m = 2$. $\blacksquare$

One can show that

$$\mathbf{A}\mathbf{A}^+ = \mathbf{U}\mathbf{U}^T$$

is an orthogonal projector onto $\mathcal{R}(\mathbf{U}) = \mathcal{R}(\mathbf{A})$, while

$$\mathbf{A}^+\mathbf{A} = \mathbf{V}\mathbf{V}^T$$

is an orthogonal projector onto $\mathcal{R}(\mathbf{V}) = \mathcal{R}(\mathbf{A}^T)$.

Let us wrap up this section by collecting the results for the problem that motivated us.

Theorem 7.1. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be a rank-r matrix with (reduced and dyadic) SVD*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

Consider the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|.$$

- *If $r = n$, the least squares problem has a unique solution, given by*

$$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b} = \sum_{j=1}^{r} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j. \tag{7.6}$$

- *If $r < n$, the least squares problem has infinitely many solutions. Any vector of the form*

$$\mathbf{x} = \sum_{j=1}^{r} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j + \sum_{j=r+1}^{n} d_{j-r}\mathbf{v}_j$$

minimizes $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ for any choice of $d_1, \ldots, d_{n-r} \in \mathbb{R}$. Among these infinitely many solutions,

$$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b} = \sum_{j=1}^{r} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j \tag{7.7}$$

Notice that the expressions for $\mathbf{x}_+$ given in (7.6) and (7.7) are identical.

7.3 *An introduction to ill-posed problems*

The pseudoinverse gives us the ultimate solution to linear systems and least squares problems. But just before we begin to rest on our laurels, we should look at one last example that will take a bit of the shine off $\mathbf{A}^+$.

Example 7.8. (Sensitivity of the pseudoinverse) Recall Example 7.4,

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

with infinitely many solutions

$$\mathbf{x} = \begin{bmatrix} x_1 \\ 1 - x_1 \end{bmatrix}$$

from which we pick out the minimum norm solution

$$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b} = \begin{bmatrix} 1/10 & 2/10 \\ 1/10 & 2/10 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}.$$

Suppose we tweak $(2,2)$ entry of $\mathbf{A}$ with a small perturbation ("noise"):

Of course, application problems always feature some kind of *noise*. At a minimum, some noise stems from floating point rounding error; in many settings, such arithmetic noise is dwarfed by more significant measurement errors.

$$\mathbf{A}_\varepsilon = \begin{bmatrix} 1 & 1 \\ 2 & 2+\varepsilon \end{bmatrix}.$$

where $\varepsilon > 0$ is some small number. This change makes $\mathbf{A}_\varepsilon$ invertible:

$$\mathbf{A}_\varepsilon^+ = \mathbf{A}_\varepsilon^{-1} = \begin{bmatrix} 2/\varepsilon + 1 & -1/\varepsilon \\ -2/\varepsilon & 1/\varepsilon \end{bmatrix}.$$

Notice that small ε give very large entries in this matrix; for $\varepsilon = 0.01$, we have

$$\mathbf{A}_{.01}^+ = \begin{bmatrix} 201 & -100 \\ -200 & 100 \end{bmatrix},$$

while for $\varepsilon = 0.0001$,

$$\mathbf{A}_{.0001}^+ = \begin{bmatrix} 20001 & -10000 \\ -20000 & 10000 \end{bmatrix}.$$

The smaller ε gets, the larger the entries in the pseudoinverse!

In particular, we see that even though $\mathbf{A}_\varepsilon \to \mathbf{A}$ as $\varepsilon \to 0$,

$$\mathbf{A}_\varepsilon^+ \text{ does not converge to } \mathbf{A}^+ \text{ as } \varepsilon \to 0.$$

Apparently the pseudoinverse is *not a continuous function* of the matrix entries.

To examine what is going on here in a (relatively) clean manner, we will switch a more strategic perturbation. Now we will slightly disturb all entries of $\mathbf{A}$ and $\mathbf{b}$:

We could have done the same analysis for the $\mathbf{A}_\varepsilon$ given above, with very similar results, but the SVD would be more complicated.

$$\mathbf{A}_\varepsilon = \begin{bmatrix} 1+2\varepsilon & 1-2\varepsilon \\ 2-\varepsilon & 2+\varepsilon \end{bmatrix}, \qquad \mathbf{b}_\varepsilon = \begin{bmatrix} 1+3\varepsilon \\ 2+\varepsilon \end{bmatrix}.$$

Notice that all of these changes are still small when $\varepsilon > 0$ is small. As before, $\mathbf{A}_\varepsilon$ is invertible, with

$$\begin{aligned} \mathbf{A}_\varepsilon^+ = \mathbf{A}_\varepsilon^{-1} &= \frac{1}{10\varepsilon} \begin{bmatrix} 2+\varepsilon & -1+2\varepsilon \\ -2+\varepsilon & 1+2\varepsilon \end{bmatrix} \\ &= \begin{bmatrix} 2/(10\varepsilon) & -1/(10\varepsilon) \\ -2/(10\varepsilon) & 1/(10\varepsilon) \end{bmatrix} + \begin{bmatrix} 1/10 & 2/10 \\ 1/10 & 2/10 \end{bmatrix}. \end{aligned} \tag{7.8}$$

This last decomposition is very suggestive: $\mathbf{A}_\varepsilon^+$ consists of a large piece that explodes when $\varepsilon \to 0$ (the first matrix) followed by a constant matrix independent of ε that actually equals the pseudoinverse

of the original (unperturbed) matrix, $\mathbf{A}^+$. In a sense, the piece of the pseudoinverse that we want is swamped by the term that comes from the noise.

For deeper insight, let us (naturally) turn to the SVD of $\mathbf{A}_\varepsilon$. The dyadic form is

$$\mathbf{A}_\varepsilon = \sqrt{10}\begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} + \varepsilon\sqrt{10}\begin{bmatrix} 2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix},$$

with singular values $\sigma_1 = \sqrt{10}$ and $\sigma_2 = \varepsilon\sqrt{10}$. The first term

$$\sigma_1\mathbf{u}_1\mathbf{v}_1^T = \sqrt{10}\begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

is precisely the unperturbed matrix $\mathbf{A}$ (see the SVD in Example 7.7), while the second term

$$\sigma_2\mathbf{u}_2\mathbf{v}_2^T = \varepsilon\sqrt{10}\begin{bmatrix} 2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix},$$

is purely attributable to the noise. No problem: as $\varepsilon \to 0$ this term becomes increasingly insignificant.

The perturbation was carefully rigged so these two terms separate so nicely, to ease the presentation. In most cases, the first term will have a small contribution from ε that goes away as $\varepsilon \to 0$, but the phenomena discussed here occurs essentially the same; the math is just messier.

With the SVD in hand, we can readily compute the pseudoinverse:

$$\begin{aligned}\mathbf{A}_\varepsilon^+ &= \frac{1}{\sigma_1}\mathbf{v}_1\mathbf{u}_1^T + \frac{1}{\sigma_2}\mathbf{v}_2\mathbf{u}_2^T \\ &= \frac{1}{\sqrt{10}}\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}\begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix} + \frac{1}{\varepsilon\sqrt{10}}\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}\begin{bmatrix} 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix} \\ &= \begin{bmatrix} 1/10 & 2/10 \\ 1/10 & 2/10 \end{bmatrix} + \frac{1}{\varepsilon}\begin{bmatrix} 2/10 & -1/10 \\ -2/10 & 1/10 \end{bmatrix}.\end{aligned}$$

The second term, which vanished as $\varepsilon \to 0$ in the expression for $\mathbf{A}_\varepsilon$ since $\sigma_2 \to 0$, now blows up like $1/\varepsilon$ in $\mathbf{A}_\varepsilon^+$ because of $1/\sigma_2 \to \infty$.

Pause for a moment to appreciate the importance of this observation, which reveals a major blemish on the heretofore pristine pseudoinverse and will occupy our attention in the rest of this chapter.

What implications does this property of the pseudoinverse have for the solution of linear systems? We want to solve $\mathbf{A}_\varepsilon\mathbf{x}_\varepsilon = \mathbf{b}_\varepsilon$ for $\mathbf{x}_\varepsilon$. Since $\mathbf{A}_\varepsilon$ is invertible, we can use the formula in (7.8) to get

$$\mathbf{x}_\varepsilon = \mathbf{A}_\varepsilon^+\mathbf{b}_\varepsilon = \mathbf{A}_\varepsilon^{-1}\mathbf{b}_\varepsilon = \frac{1}{10\varepsilon}\begin{bmatrix} 2+\varepsilon & -1+2\varepsilon \\ -2+\varepsilon & 1+2\varepsilon \end{bmatrix}\begin{bmatrix} 1+3\varepsilon \\ 2+\varepsilon \end{bmatrix} = \begin{bmatrix} 1+\varepsilon/2 \\ \varepsilon/2 \end{bmatrix}.$$

Recall that $\mathbf{x}_+ = \mathbf{A}^+\mathbf{b}$ was the minimum norm solution of the least squares problem,

$$\mathbf{x}_+ = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}.$$

Adding the perturbation gives the solution $\mathbf{x}_\varepsilon$ an entirely different character:

$$\mathbf{x}_\varepsilon = \begin{bmatrix} 1+\varepsilon/2 \\ \varepsilon/2 \end{bmatrix} \to \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The figures on the right illustrates this difference. The solution $\mathbf{x}_\varepsilon$ to the perturbed solution is converging to *a solution* of the original problem, but it is far from the minimum-norm solution $\mathbf{x}_+ = \mathbf{A}^+\mathbf{b}$ given by the pseudoinverse of $\mathbf{A}$.

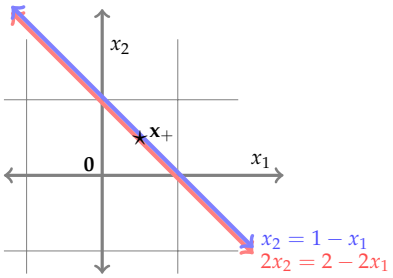
The situation can even be worse than shown here. Suppose we keep $\mathbf{A}_\varepsilon$ the same but adjust $\mathbf{b}_\varepsilon$ strategically

$$\mathbf{b}_\varepsilon = \begin{bmatrix} 1+2\sqrt{\varepsilon} \\ 2-\sqrt{\varepsilon} \end{bmatrix} = \mathbf{b} + \sqrt{5\varepsilon}\mathbf{u}_2$$

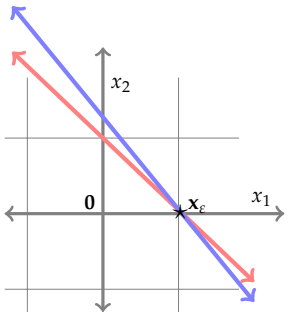
with noise in the direction $\mathbf{u}_2$, the left singular vector associated with the small singular value $\sigma_2 = \varepsilon\sqrt{10}$. Now the (unique) solution to the perturbed system is

$$\mathbf{x}_\varepsilon = \begin{bmatrix} 1/2+1/(2\sqrt{\varepsilon}) \\ 1/2-1/(2\sqrt{\varepsilon}) \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} + \frac{1}{\sqrt{\varepsilon}} \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix}. \tag{7.9}$$

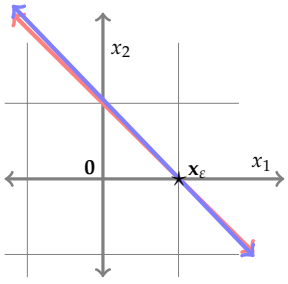
Take a moment to consider this solution. What happens as $\varepsilon \to 0$? Unlike the first perturbation to $\mathbf{b}_\varepsilon$, *the solution now grows extremely large, simply because of the noise.*



The original problem $\mathbf{Ax} = \mathbf{b}$, has infinitely many solutions with $x_2 = 1 - x_1$, with the pseudoinverse solution $\mathbf{x}_+ = [1/2, 1/2]^T$ minimizing $\|\mathbf{x}\|$.



The perturbed problem $\mathbf{A}_\varepsilon\mathbf{x}_\varepsilon = \mathbf{b}_\varepsilon$ has a unique solution, $\mathbf{x}_\varepsilon = [1+\varepsilon/2, \varepsilon/2]^T$, shown here with $\varepsilon = 0.05$.



Repetition of the $\mathbf{A}_\varepsilon\mathbf{x}_\varepsilon = \mathbf{b}_\varepsilon$, but now with $\varepsilon = 0.01$. The line becomes closer but only intersect at one point $\mathbf{x}_\varepsilon$, quite distant from the pseudoinverse solution $\mathbf{x}_+$ to the original problem.

7.4 *Statistics of least squares*

7.5 *Application: deblurring*

The examples presented thus far seem quite sterile, cooked up to exhibit pathology. We study this topic because it has tremendous importance in applications. The general field is known as *inverse problems*; prominent applications include all sorts of image processing problems, from sharpening up astronomical images to detecting tumor margins from medical scans.

Let us introduce these ideas with a simple example of deblurring a one dimensional "image," like the barcode discussed in Chapter 1. To set the stage, we need a mathematical mechanism to *blur* an image. Our "image" will be a function of one variable, $f(s)$ for $s \in [0,1]$. To blur the image, we "*convolve* the image with a *kernel* function." This is a fancy way of saying that we integrate the product of f against some other function h:

$$\int_0^1 h(s,t)f(t)\,dt = b(s). \tag{7.10}$$

We call $h(s,t)$ the *blurring kernel*.

For the barcode example, $f(s) = 1$ will correspond to a black bar, while $f(s) = 0$ corresponds to a white bar. As s goes from $s = 0$ to $s = 1$, $f(s)$ jumps from between 0 and 1: it has *jump discontinuities* at every transition from a black to white bar, and vice versa. See page 3 in Chapter 1 for an illustration.

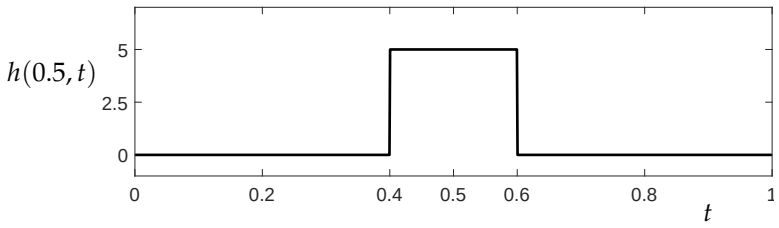


Figure 7.3: The step function blurring kernel $h(s,t)$ given by (7.11) with $z = 0.1$, shown at $s = 0.5$ for $t \in [0,1]$. On the interval $[s-z, s+z]$ the function takes the value $1/(2z) = 5$.

Let us take a simple concrete example: define

$$h(s,t) = \begin{cases} 1/(2z), & |s-t| \le z; \\ 0, & |s-t| > z. \end{cases} \tag{7.11}$$

The parameter $z > 0$ determines the nature of the blur. Figure 7.3 shows this kernel evaluated at the point $s = 0.5$ for $z = 0.1$. Decreasing z will cause $h(s,t)$ to be nonzero over a narrower interval in t centered at s (but taller there, $1/(2z)$).

Fix a point $s \in [0,1]$. Away from the edges of the domain (i.e., when $z \le s \le 1-z$, this kernel (7.11) causes the integral (7.10) to take the form

$$\begin{aligned} b(s) = \int_0^1 h(s,t) f(t)\,\mathrm{d}t &= \int_{s-z}^{s+z} \left(\frac{1}{2z}\right) f(t)\,\mathrm{d}t \\ &= \frac{\int_{s-z}^{s+z} f(t)\,\mathrm{d}t}{2z} \\ &= \frac{\text{area under } f \text{ over } [s-z, s+z]}{\text{width of interval } [s-z, s+z]}. \end{aligned}$$

Thus the integral is a *moving average* of f taken over a window of width $2z$ centered at s. You can thus see how the integral in (7.10) smooths, or *blurs* f: fine-scale changes in f are smeared out over the averaging process.

In imaging applications, we *measure* the blurry $b(s)$ function (with a scanner, camera, telescope, etc.) and try to *solve* equation (7.10) for the *unknown* $f(t)$, the original object we seek. Thus solution process should undo the blurring operation, that is, it should *sharpen* up the image. However, we might be suspicious that this process could be delicate. Two distinct objects might differ in fine-scale detail, but this detail gets lost in the blur: *the blurring operation maps two different objects $f_1 \neq f_2$ to very similar images $b_1 \approx b_2$*. The rest of this section will explore how this potential sensitivity plays out in linear algebra.

7.5.1 Discretization turns calculus into linear algebra

Modern cameras store images digital information, recording color intensity at n *pixel* values: we must cast the function $b(s)$ for $s \in [0,1]$ into a vector $\mathbf{b}$ of n discrete values. To convert the calculus problem (7.10) into a linear algebra problem, break the interval $[0,1]$ into discrete points:

$$s_j = \frac{j-1/2}{n}, \qquad t_k = \frac{k-1/2}{n}, \qquad j,k = 1,\ldots,n,$$

for some $n \geq 1$. Then approximate the integral (7.10) by the simple *midpoint quadrature rule*:

$$\frac{1}{n}\sum_{k=1}^{n} h(s_j,t_k)f_k = b_j, \qquad j = 1,\ldots,n, \quad (7.12)$$

where $b_j = b(s_j)$ and (hopefully) $f_k \approx f(t_k)$.

Arranging all n of the equations (7.12) for $j = 1,\ldots,n$ into matrix form gives $\mathbf{A}\mathbf{f} = \mathbf{b}$:

$$\frac{1}{n}\begin{bmatrix} h(s_1,t_1) & h(s_1,t_2) & \cdots & h(s_1,t_n) \\ h(s_2,t_1) & h(s_2,t_2) & \cdots & h(s_2,t_n) \\ \vdots & \vdots & \ddots & \vdots \\ h(s_n,t_1) & h(s_n,t_2) & \cdots & h(s_n,t_n) \end{bmatrix}\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \quad (7.13)$$

Be sure to include the $1/n$ term in the matrix $\mathbf{A}$: that term is crucial to get the scaling right.

7.5.2 An example

The kernel (7.11) illustrated in Figure 7.3 is a bit too crude. In many applications, one prefers *Gaussian kernels* of the form

$$h(s,t) = \frac{1}{\sqrt{\pi}z}\mathrm{e}^{-(s-t)^2/z^2}.$$

Scaling by $1/(\sqrt{\pi}z)$ ensures that $\int_{-\infty}^{\infty} h(s,t)\,\mathrm{d}t = 1$. We are only integrating from over $t \in [0,1]$, but the tails of $h(s,t)$ that we neglect would make a very small contribution when z is small and s is not near the edges of the domain $[0,1]$.

for some small constant $z > 0$. When $s \approx t$, $h(s,t) \approx 1/(\sqrt{\pi}z)$; when $|s-t|$ is large, $h(s,t) \approx 0$. For illustrative purposes here will shall use a simpler kernel, the "hat function" shown in Figure 7.4,

$$h(s,t) = \max\Big(0, 1-|s-t|/z\Big)/z, \quad (7.14)$$

whose behavior is similar to the Gaussian kernel.

For this experiment, we will use $n = 1000$ discretization points over the interval $s,t \in [0,1]$. First we will form a known $\mathbf{f}$ vector and show how the operation $\mathbf{b} = \mathbf{A}\mathbf{f}$ does indeed *blur* $\mathbf{f}$. Take as the original object $f(t)$ the function plotted in Figure 7.5. We have given it a few sharp corners ("edges" in image processing) that we expect a

blurring operation to smooth out. Construct the vector $\mathbf{f} \in \mathbb{R}^{1000}$ by sampling $f(t_k)$ at the points $t_k = (k - 1/2)/n$ for $k = 1, \ldots, n = 1000$, and form the blurring matrix $\mathbf{A} \in \mathbb{R}^{1000 \times 1000}$ for the hat kernel (7.14) with $z = 0.05$, using the structure given in (7.13).

Figure 7.6 compares f to the entries in $\mathbf{b} = \mathbf{A}\mathbf{f}$. The vector $\mathbf{b} \in \mathbb{R}^{1000}$ appears as red dots (fused into a continuous curve) with t_k on the horizontal axis and the kth entry of $\mathbf{b}$ on the vertical axis, to facilitate comparison with $f(t)$ over $t \in [0, 1]$. Indeed, the blurred vector $\mathbf{b}$ smooths out the corners in f.

We now return to a concern we raised a few pages earlier. Can the blurring operation take two functions, say f and $\widetilde{f}$, and blur them into vectors $\mathbf{b}$ and $\widetilde{\mathbf{b}}$ that are very close together, as the cartoon in Figure 7.7 suggests? (This property would have serious implications for the *deblurring* process, which attempts to compute $\mathbf{f}$ as $\mathbf{A}^{-1}\mathbf{b}$, as we will discuss momentarily.)

Figure 7.8 confirms that the scenario envisioned in Figure 7.7 can indeed occur in practice. The left of Figure 7.8 shows the vector $\mathbf{f}$ sampled from the function f in Figure 7.5, along with a second sampled vector $\widetilde{\mathbf{f}}$. This new vector $\widetilde{\mathbf{f}}$ differs considerably from $\mathbf{f}$, but the changes (bottom of Figure 7.8) are fine in scale; the overall shape of $\mathbf{f}$ and $\widetilde{\mathbf{f}}$ is the same. (Because of its many oscillations, we say the difference is *high in frequency*.)

Now compare the blurred versions of $\mathbf{f}$ and $\widetilde{\mathbf{f}}$, denoted by $\mathbf{b}$ and $\widetilde{\mathbf{b}}$ and shown on the right side of Figure 7.8. To the eye, *it is very difficult to tell any difference between* $\mathbf{b}$ *and* $\widetilde{\mathbf{b}}$ despite the significant difference between $\mathbf{f}$ and $\widetilde{\mathbf{f}}$. The bottom-right plot confirms this: even though $\widetilde{\mathbf{f}}$ differs from $\mathbf{f}$ by about 0.1, $\widetilde{\mathbf{b}}$ never differs from $\mathbf{b}$ by more than 0.0001, *three orders of magnitude less!*

7.5.3 *Deblurring: a prototypical inverse problem*

Thus far our example has taken a known object $\mathbf{f}$, and created a blurry version $\mathbf{b}$. Rarely to we want to blur out an image; more often we want to take a blurring image acquired from our "camera" and sharpen it up.

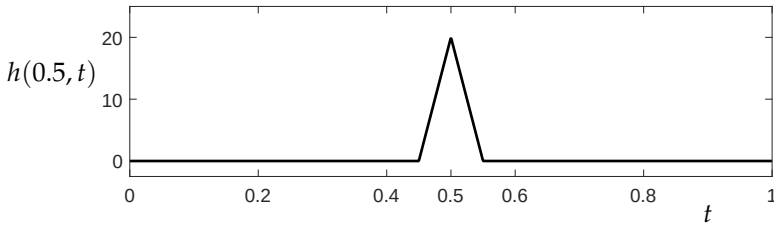


Figure 7.4: The hat function blurring kernel $h(s, t)$ given by (7.14) with $z = 0.05$, shown at $s = 0.5$ for $t \in [0, 1]$.

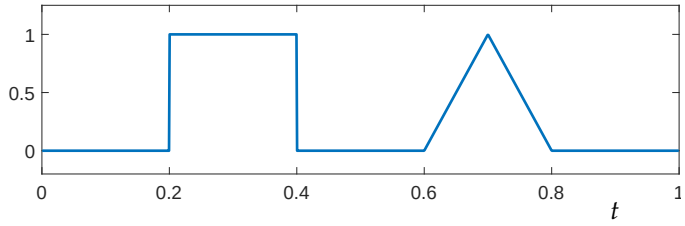


Figure 7.5: The original "object" described by the function f.

We hope to discover f by deblurring the acquired image b.

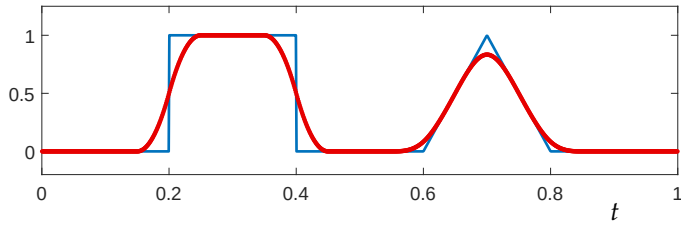


Figure 7.6: The function f (in blue), compared with samples of the blurred vector $\mathbf{b} = \mathbf{A}\mathbf{f}$ (in red), using the hat function kernel (7.14) with $z = 0.05$, discretized with $n = 1000$ points.

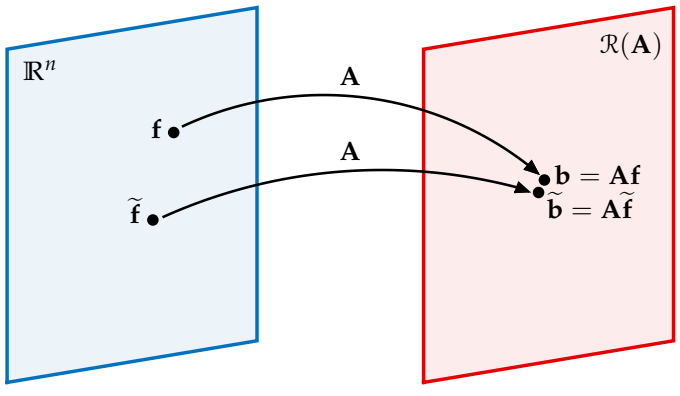
To deblur the image $\mathbf{b}$, we could simply compute

$$\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}.$$

With the kernel (7.14), the matrix $\mathbf{A}$ is invertible, so $\mathbf{A}^{+} = \mathbf{A}^{-1}$.

Figure 7.9 sketches out this process and highlights a major challenge: by smoothing out fine-scale details, the blurring operation can map quite different $\mathbf{f}$ vectors to very similar $\mathbf{b}$ vectors, as seen in Figure 7.8. The blurring operation has essentially erased the fine-level detail in $\widetilde{Bf}$. This property is highly problematic when we reverse the direction of the map in the inverse problem: *small changes in the acquired image* $\mathbf{b}$ *can result in very different sharpened images* $\mathbf{f}$*!* You can imagine the kinds of noise that could make small changes to the $\mathbf{b}$ vector acquired by our "camera." A process like this is called an "ill-posed inverse problem," because the process of undoing the blurring can effectively result in many different reasonable answers.

Figure 7.10 explore the practical implications of this *ill-posedness*.



FORWARD PROBLEM

Figure 7.7: Cartoon of the *blurring* process: the matrix $\mathbf{A}$ can map vectors that are far apart to vectors that are quite close together. We call this the "forward problem": $\mathbf{A}$ maps a known vector $\mathbf{f}$ to its blurred version $\mathbf{b} = \mathbf{A}\mathbf{f}$.

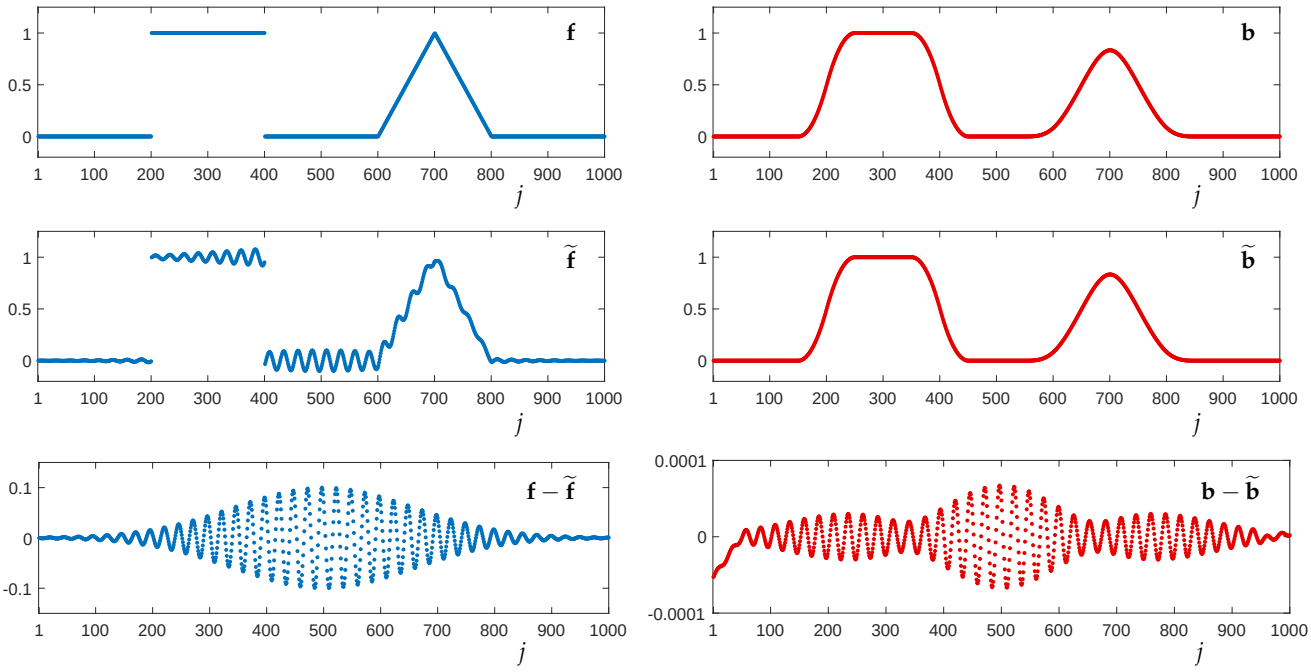


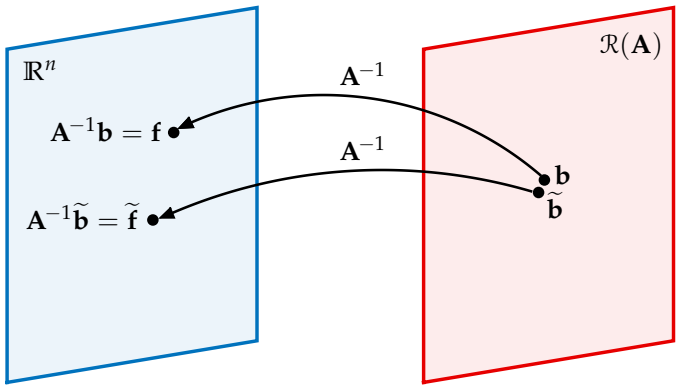
Figure 7.8: Blurring maps vectors closer together. Consider the two vectors $\mathbf{f}$ and $\widetilde{\mathbf{f}}$ on the left, whose difference, which is obvious to the eye, is plotted in the bottom left. Despite the differences between $\mathbf{f}$ and $\widetilde{\mathbf{f}}$, their blurred versions $\mathbf{b} = \mathbf{A}\mathbf{f}$ and $\widetilde{\mathbf{b}} = \mathbf{A}\widetilde{\mathbf{f}}$ (shown on the right) *look nearly identical*. Their difference, on the bottom right, does not exceed 0.0001.

Start with the top-left plot, which deblurs the exact vector $\mathbf{b}$ via $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$, showing the result as yellow dots. (The true $f(t)$ function is shown in blue underneath the yellow dots, for reference.) The recovered $\mathbf{f}$ looks perfect to the eye, because we knew the exact function $\mathbf{b}$.

Suppose now that we inject a bit of noise in the process, and follow this testing procedure.

- Take the exact solution $\mathbf{f}$ with kth entry $f_k = f(t_k)$.
- Form the exact right-hand side $\mathbf{b} = \mathbf{A}\mathbf{f}$ by multiplying $\mathbf{A}$ against $\mathbf{f}$.

In some settings – even using different kernels in this one-dimensional deblurring process – the inversion process is so fragile that calculating $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$ results in a ridiculous answer.



INVERSE PROBLEM

Figure 7.9: Cartoon of the *deblurring* process. We call this an "inverse problem" because it takes an acquired image $\mathbf{b}$ and seeks to "invert" the blurring process to get the original (unknown) sharp image $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$. Deblurring is the prototypical example of an *ill-posed problem*. Noise makes small changes to $\mathbf{b}$, which can get highly exaggerated by the action of $\mathbf{A}^{-1}$: *the inverse can map nearby vectors* $\mathbf{b}$ *and* $\widetilde{\mathbf{b}}$ *to far off points* $\mathbf{f}$ *and* $\widetilde{\mathbf{f}}$.

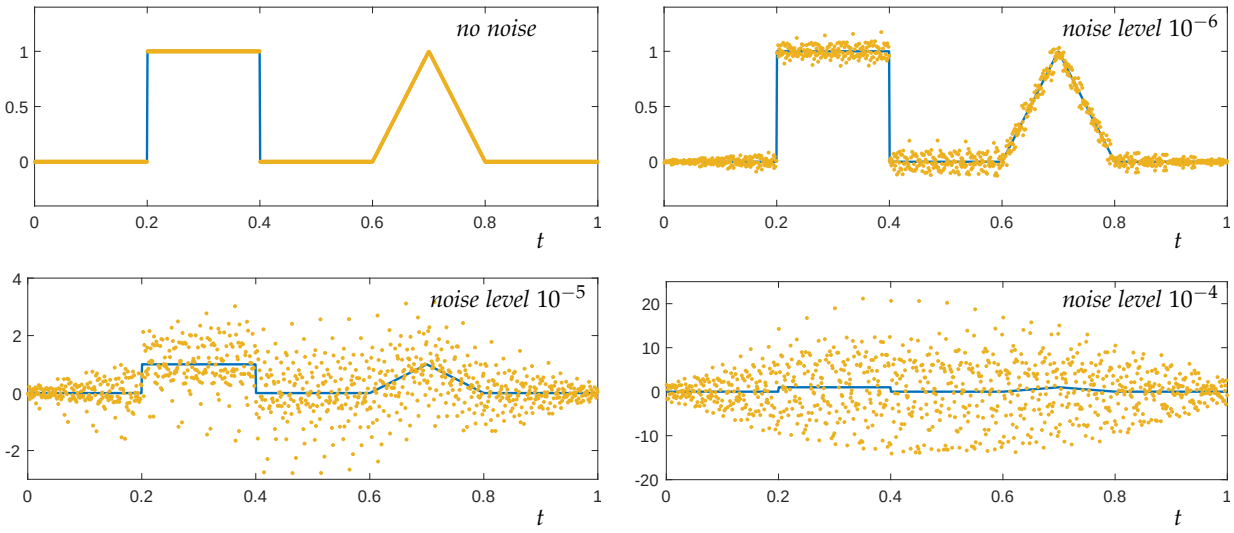


Figure 7.10: Recovered functions $\mathbf{f}_{\text{noise}} = \mathbf{A}^{-1}\mathbf{b}_{\text{noise}}$. The noise level refers to the standard deviation of random noise added to the original $\mathbf{b}$ vector.

- Add Gaussian noise to the right-hand side,

$$\mathbf{b}_{\text{noise}} = \mathbf{b} + \boldsymbol{\varepsilon},$$

  where the entries of $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ are Gaussian random variables with mean 0 and small standard deviation. (The experiments in Figure 7.10 use standard deviation 10^{-6}, 10^{-5}, and 10^{-4}.)

- Try to recover the exact $\mathbf{f}$ from the noisy data,

$$\mathbf{f}_{\text{noise}} = \mathbf{A}^{-1}\mathbf{b}_{\text{noise}}.$$

Note that we compute all "$\mathbf{A}^{-1}\mathbf{b}$" type operations by solving a linear system, which you can do via the `np.linalg.solve` command. You should not form the matrix $\mathbf{A}^{-1}$.

Figure 7.10 shows the results of three trials of this experiment with different noise levels. Even with noise level as small as 10^{-6} the results are pretty shabby; indeed, the recovered $\mathbf{f}_{\text{noise}}$ resembles the $\widehat{\mathbf{f}}$ vector in Figure 7.8. Increase the noise level by an order of magnitude or two and *the results are entirely useless*!

7.5.4 *The rise of the small singular values*

To understand the results in Figure 7.10, let us revisit the inversion operation. Please note that the discussion here goes through exactly the same if $\mathbf{A}$ is not invertible and we are discussing instead the pseudoinverse solution $\mathbf{A}^{+}\mathbf{b}$. In this spirit, we continue the discussion in terms of a rank-r matrix, even though our running example, our particular blurring matrix has rank $r = n$.

Consider the standard $\mathbf{Af} = \mathbf{b}$ problem. The inverse (or pseudoinverse) solution is given by

$$\mathbf{f} = \sum_{j=1}^{r} \frac{\mathbf{u}_j^T \mathbf{b}}{\sigma_j} \mathbf{v}_j. \tag{7.15}$$

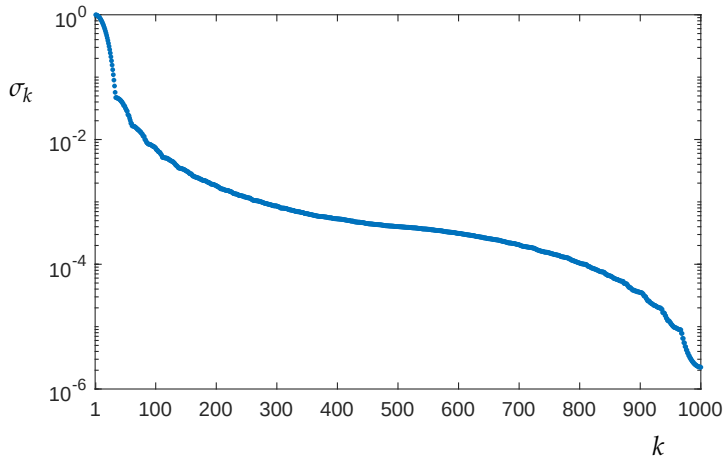


Figure 7.11: Singular values of the 1000×1000 blurring matrix with the hat function kernel shown in Figure 7.4.

The σ_j values in the denominator will be our primary concern: when we invert singular values, the small σ_j terms become the dominant $1/\sigma_j$ terms! If the corresponding value of $\mathbf{u}_j^T\mathbf{b}$ is not correspondingly small, these terms will come to dominate $\mathbf{f}$.

Let us inspect the singular values of the blurring matrix $\mathbf{A}$, shown in Figure 7.11. Notice that the singular values decay rapidly, a drop of *six orders of magnitude* from $\sigma_1 \approx 1$ to $\sigma_{1000} \approx 2 \times 10^{-6}$. Thus we need to be concerned about the role of small singular values in the solution formula (7.15).

Note that when $\mathbf{b} \in \mathcal{R}(\mathbf{A})$, we can write

$$\mathbf{b} = \mathbf{U}\mathbf{U}^T\mathbf{b} = \sum_{j=1}^{r} (\mathbf{u}_j^T\mathbf{b})\mathbf{u}_j. \tag{7.16}$$

The numbers $\mathbf{u}_j^T\mathbf{b}$ are the coefficients of $\mathbf{b}$ expanded in the basis $\{\mathbf{u}_j\}$ for $\mathcal{R}(\mathbf{A})$.

This expansion leads to an observation that will be very important for our discussion going forward.

The scalars $\mathbf{u}_j^T\mathbf{b} \in \mathbb{R}$ tell us how much the jth left singular vector $\mathbf{u}_j$ contributes to $\mathbf{b}$.

Now using the formula $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T$ derived in equation (7.7), we have

$$\mathbf{f} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \sum_{j=1}^{r} \left(\frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\right)\mathbf{v}_j. \tag{7.17}$$

The numbers $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ are the coefficients of $\mathbf{f}$ expanded in the basis $\{\mathbf{v}_j\}$ for $\mathcal{R}(\mathbf{A}^T)$.

In most situations the true object $\mathbf{f}$ is modest in size, say $\|\mathbf{f}\| \approx 1$. Since the right singular vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$ are orthonormal,

$$\|\mathbf{f}\|^2 = \mathbf{f}^T\mathbf{f} = \left(\sum_{j=1}^{r} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j\right)^T \left(\sum_{k=1}^{r} \frac{\mathbf{u}_k^T\mathbf{b}}{\sigma_k}\mathbf{v}_k\right)$$

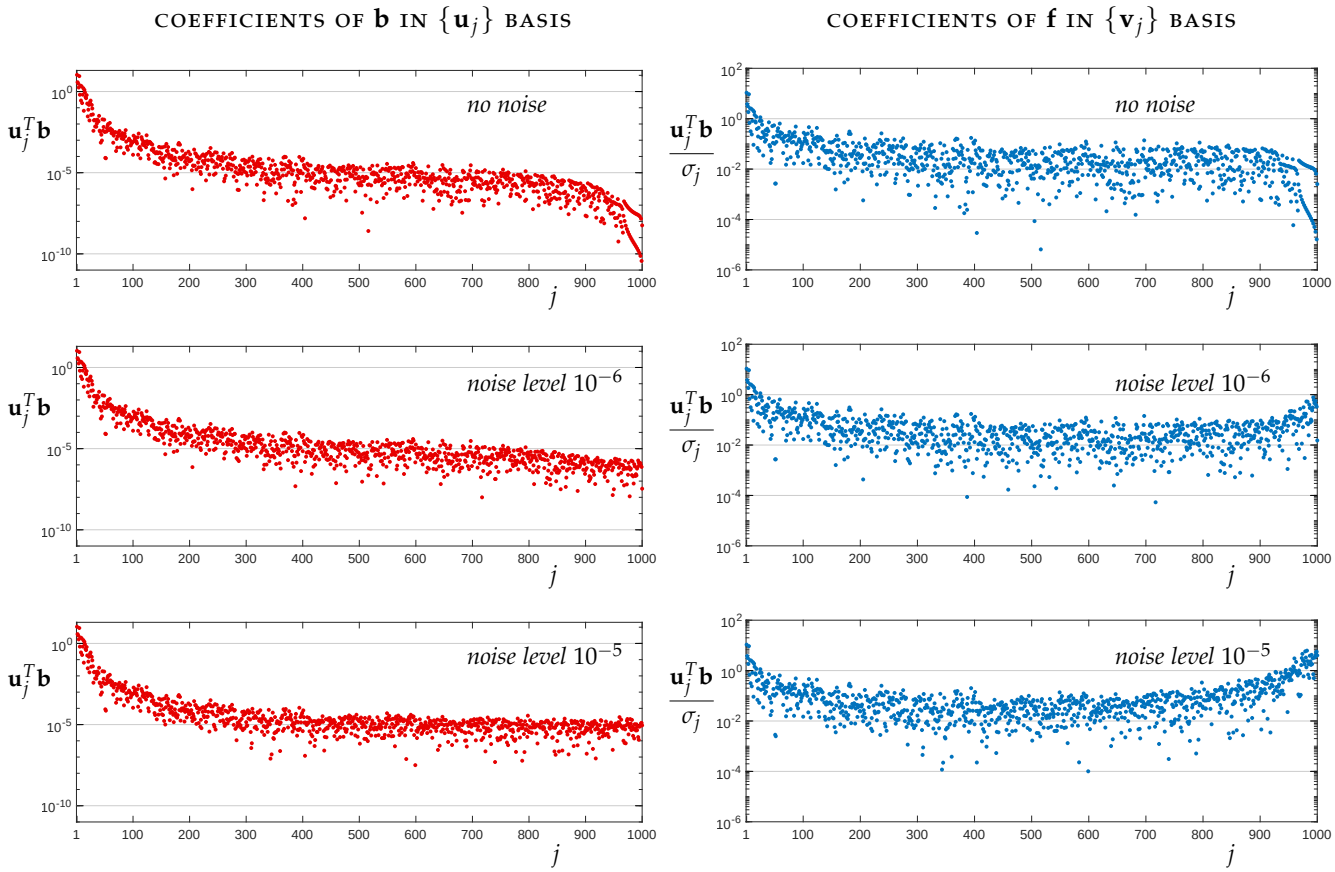


Figure 7.12: Coefficients of $\mathbf{b}$ and $\mathbf{f}$, for three experiments from Figure 7.10. The top plots use $\mathbf{b}$ and $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$; the middle plots use $\mathbf{b}_{\text{noise}}$ with standard deviation 10^{-6}; the bottom plots again use $\mathbf{b}_{\text{noise}}$ but with standard deviation 10^{-5}. Take special note of two features as the noise increases from top to bottom: (1) the noise impedes the decay of the last coefficients $\mathbf{u}_j^T\mathbf{b}$ on the left; (2) as a consequence, for the noisy plots on the right we see that last coefficients $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ are on a similar level as the leading coefficients, significantly polluting the results seen in Figure 7.10.

$$= \sum_{j=1}^{r}\sum_{k=1}^{r}\left(\frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\right)\left(\frac{\mathbf{u}_k^T\mathbf{b}}{\sigma_k}\right)\mathbf{v}_j^T\mathbf{v}_k = \sum_{j=1}^{r}\left(\frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\right)^2.$$

From this calculation we make an essential observation.

If the solution $\mathbf{f}$ is not large, say $\|\mathbf{f}\| \approx 1$, and the singular value σ_j is small, then the corresponding value of $\mathbf{u}_j^T\mathbf{b}$ must also be small, to keep $(\mathbf{u}_j^T\mathbf{b}/\sigma_j)^2$ from making too large a contribution to $\|\mathbf{f}\|^2 \approx 1$.

Now we are prepared to make sense of the results in Figure 7.10. Figure 7.12 reveals everything; study these plots carefully!

- In the absence of noise (top plots), the coefficients $\mathbf{u}_j^T\mathbf{b}$ decay overall quite steadily as j increases, with a big dip in values for the last 50 values of j. (This dip corresponds to an abrupt drop in the final singular values of $\mathbf{A}$ in Figure 7.11.) When scaled by σ_j to give the coefficients of $\mathbf{f}$ on the right, we also see an overall drop as j increases. In particular, for the largest values of j, $\mathbf{v}_j$ makes little contribution to $\mathbf{f}$.
- When noise is added (middle and bottom plots), the values of $\mathbf{u}_j^T\mathbf{b}$

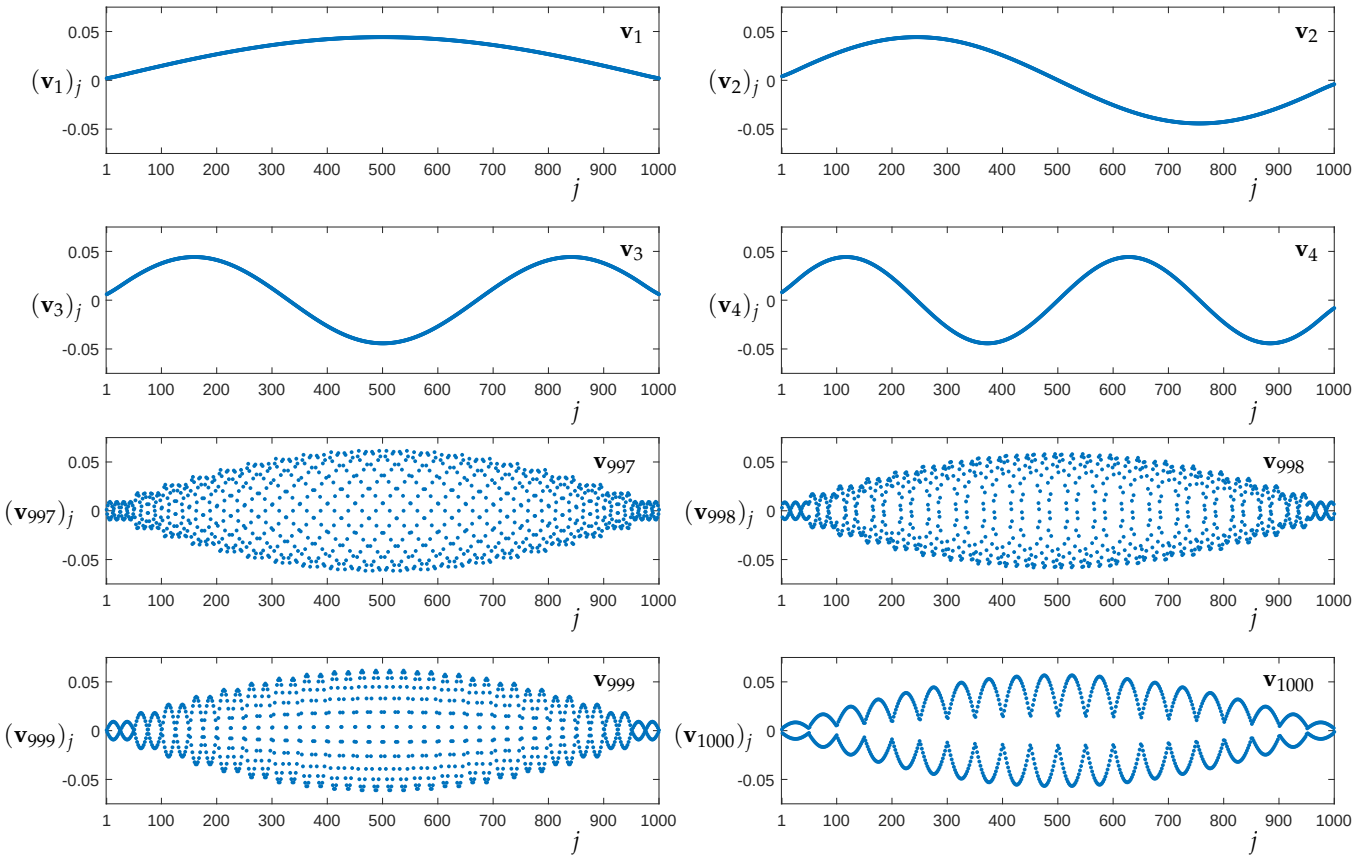


Figure 7.13: Right singular vectors $\mathbf{v}_k$ of the blurring matrix $\mathbf{A}$ for $n = 1000$ and with the kernel shown in Figure 7.4. (Each plot shows 1000 blue dots, which are sufficiently close together to look like smooth curves in the top plots.) The first four singular vectors (top) are *low frequency*: the entries oscillate very gradually over the 1000 entries. In contrast, the last four singular vectors are *high frequency*, oscillating wildly.

change quite a bit, especially for $j > 900$. *The rapid drop for large j seen in the noiseless case has gone away.* As a consequence, on the right we see large values of $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ when j is large.

- For noise level 10^{-6}, the trailing values of $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ are still smaller than the values for small j, and hence we can still see the correct overall shape for $\mathbf{f}$ in Figure 7.10.
- For noise level 10^{-5}, values of $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ for large j dominate those for small j, and hence in Figure 7.10 the recovered $\mathbf{f}$ has lost most of the shape of the true solution.
- Look again at that top-left plot. The bottom coefficient $\mathbf{u}_{1000}^T\mathbf{b}$ is quite small; if we add 10^{-5} to it, we have only made a perturbation to $\mathbf{b}$ of that same size. However, when we divide that perturbed value of $\mathbf{u}_{1000}^T\mathbf{b}$ by σ_{1000}, suddenly we have overwhelmed all the other coefficients that make up $\mathbf{f}$.

One final wrinkle will complete our understanding.

In the noisiest examples shown in Figure 7.10, why do the recovered values of $\mathbf{f}$ have that particular "football" shape?

Figure 7.13 shows eight right singular vectors $\mathbf{v}_j$ of $\mathbf{A}$, corresponding to the four largest and four smallest singular values.

Look first at those top four plots: the dots we plot fuse together into lines that resemble sine functions of increasing frequency; they are very smooth. If our $\mathbf{f}$ is a smooth function, it should be well-approximated by a linear combination of smooth vectors like this.

Fourier analysis gives the mathematical tools needed to make statements like this precise.

Now look at the last four singular vectors. In contrast, these are "high frequency" vectors. We can mostly see individual dots, oscillating up and down often. They also have that distinctive "football" shape, pinched at the ends. For the noisy examples in Figure 7.10, the shape of these latter $\mathbf{v}_j$ vectors dominate the combination of the early $\mathbf{v}_j$ vectors that captures the essence of the true $\mathbf{f}$.

At this stage, we have fully analyzed the blurring example, at least for our choice of blurring function. *If you understand the reason for the disappointing results in Figure 7.10, captured the essential difficulty of ill-posed linear systems.*

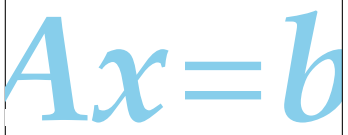
The natural next question:

Can we do anything to fix this problem?

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 8
Regularization for Ill-Posed Problems

We dedicated the last chapter to analyzing, solving, and critiquing the general least squares problem.

Least Squares problem. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Among all $\mathbf{x} \in \mathbb{R}^n$ that solve

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|, \tag{8.1}$$

find the $\mathbf{x}$ that minimizes $\|\mathbf{x}\|$.

Look back to Example 7.8 on page 104. Small perturbations of order ε to $\mathbf{A}$ and $\mathbf{b}$ caused the solution $\mathbf{x}_+$ to change by large amounts. To resolve this difficulty, your natural reaction might be: "Can't we just ignore the ε entries in $\mathbf{A}$ and $\mathbf{b}$?" This shows good instinct, but we would like a more systematic way to identify and neglect the "small" entries in a system. Thankfully the singular value decomposition provides just the right tools for the job.

This chapter presents two methods for addressing the sensitivity of the general least squares problem to small changes in $\mathbf{A}$ and $\mathbf{b}$. The first approach is natural: In the sum for the pseudoinverse solution

$$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b} = \sum_{j=1}^{r} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j,$$

just truncate the sum so that j only runs up to $k < r$, instead of all the way to r:

$$\mathbf{x}_k := \sum_{j=1}^{k} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j,$$

thus omitting the smallest singular values that cause the biggest problems, as seen in Section 7.5.4.

The second, more nuanced, approach *penalizes* the least squares objective function $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ by the norm of the solution $\|\mathbf{x}\|$, to prevent situations like those seen in equation (7.9) and the bottom-right of Figure 7.10. More precisely, given a *regularization parameter* $\lambda > 0$, we will replace (8.1) with

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda^2\|\mathbf{x}\|^2.$$

By adjusting λ we can dial-in just the right amount of penalization to give a good solution.

8.1 Regularization by truncating the SVD

The first approach summarized in (8.2) makes intuitive sense, but we would like a deeper justification for truncating the sum. Start by writing the dyadic form of the SVD of the rank-r matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$,

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

In the last chapter we showed that the pseudoinverse takes the form

$$\mathbf{A}^+ = \sum_{j=1}^{r} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T.$$

Now fix some $k \leq r$, and recall that in Section 6.2, we computed optimal rank-k approximations to $\mathbf{A}$ by truncating the SVD to the first k terms in the sum,

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T, \qquad k \leq r. \tag{8.2}$$

As we saw in Section 6.2, the accuracy of this approximation is controlled by the first neglected singular value:

$$\|\mathbf{A} - \mathbf{A}_k\| = \left\| \sum_{j=k+1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \right\| = \sigma_{k+1}. \tag{8.3}$$

The largest singular value of the error matrix $\mathbf{A} - \mathbf{A}_k$ is σ_{k+1}.

If we regard $\mathbf{A}_k$ as a good approximation of $\mathbf{A}$, then perhaps we can regard

$$\mathbf{A}_k^+ = \sum_{j=1}^{k} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T$$

Note that $\mathbf{A}_k^+$ means $(\mathbf{A}_k)^+$.

a good approximation of $\mathbf{A}^+$. On the surface this might look like a bad idea, since the error

$$\|\mathbf{A}^+ - \mathbf{A}_k^+\| = \left\| \sum_{j=1}^{r} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T - \sum_{j=1}^{k} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T \right\| = \left\| \sum_{j=k+1}^{r} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T \right\| = \frac{1}{\sigma_r}$$

could be quite large. (Note that $1/\sigma_r$ is the largest singular value of $\mathbf{A}_k^+ - \mathbf{A}_k$.) But our goal is not to *approximate* $\mathbf{A}^+$, but rather to obtain an appealing solution to the least squares problem (8.1) that is more robust to small changes in $\mathbf{A}$ and $\mathbf{b}$.

Truncating the SVD changes the fundamental subspaces by shifting vectors from $\mathcal{R}(\mathbf{A})$ to $\mathcal{N}(\mathbf{A}^T)$ and from $\mathcal{R}(\mathbf{A}^T)$ to $\mathcal{N}(\mathbf{A})$:

$$\begin{aligned}\mathcal{R}(\mathbf{A}_k) &= \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\},\\ \mathcal{N}(\mathbf{A}_k^T) &= \text{span}\{\mathbf{u}_{k+1}, \ldots, \mathbf{u}_m\},\\ \mathcal{R}(\mathbf{A}_k^T) &= \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\},\\ \mathcal{N}(\mathbf{A}_k) &= \text{span}\{\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n\}.\end{aligned}$$

Note that $\text{rank}(\mathbf{A}_k) = k$.

As usual, we use $\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m$ and $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$ to denote the extra vectors that arise when we compute the full SVD. To contrast with the box on the left, recall that

$$\begin{aligned}\mathcal{R}(\mathbf{A}) &= \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\},\\ \mathcal{N}(\mathbf{A}^T) &= \text{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\},\\ \mathcal{R}(\mathbf{A}^T) &= \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_r\},\\ \mathcal{N}(\mathbf{A}) &= \text{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}.\end{aligned}$$

Suppose we wish to solve the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|, \tag{8.4}$$

where $\mathbf{A}$ has some very small singular values that make the pseudoinverse solution derived in Section 7.2,

$$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b} = \sum_{j=1}^{r} \frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\mathbf{v}_j, \tag{8.5}$$

very large in norm. In fact, using the orthonormality of the *right* singular vectors and the Pythagorean Theorem, we can compute

$$\|\mathbf{x}_+\|^2 = \sum_{j=1}^{r} \frac{(\mathbf{u}_j^T\mathbf{b})^2}{\sigma_j^2}. \tag{8.6}$$

Let $\mathbf{y} = c_1\mathbf{q}_1 + \cdots + c_r\mathbf{q}_r$ for orthonormal vectors $\mathbf{q}_1, \ldots, \mathbf{q}_r$. Then the Pythagorean Theorem implies that

$$\begin{aligned}\|\mathbf{y}\|^2 &= \|c_1\mathbf{q}_1\|^2 + \cdots + \|c_r\mathbf{q}_r\|^2\\ &= c_1^2\|\mathbf{q}_1\|^2 + \cdots + c_r^2\|\mathbf{q}_r\|^2\\ &= c_1^2 + \cdots + c_r^2.\end{aligned}$$

While the pseudoinverse solution may lead to large $\|\mathbf{x}_+\|$, recall that is does have a virtue: it minimizes $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ over all $\mathbf{x} \in \mathbb{R}^n$. Since

$$\mathbf{b} - \mathbf{A}\mathbf{x}_+ = \sum_{j=r+1}^{m} (\mathbf{u}_j^T\mathbf{b})\mathbf{u}_j,$$

we can again use the Pythagorean Theorem, now with orthonormality of the *left* singular vectors, to see that

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_+\|^2 = \sum_{j=r+1}^{m} (\mathbf{u}_j^T\mathbf{b})^2. \tag{8.7}$$

Suppose we replace $\mathbf{A}$ in the least squares problem (8.4) with the truncated SVD $\mathbf{A}_k$,

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}_k\mathbf{x}\|. \tag{8.8}$$

The solution changes to

$$\mathbf{x}_k = \mathbf{A}_k^+ \mathbf{b} = \sum_{j=1}^{k} \frac{\mathbf{u}_j^T \mathbf{b}}{\sigma_j} \mathbf{v}_j \tag{8.9}$$

This definition for $\mathbf{x}_k$ only differs from the formula (8.5) for $\mathbf{x}_+$ in the sum's upper limit: there it was r, here it is k.

having smaller norm than $\|\mathbf{x}_+\|$, since

$$\|\mathbf{x}_k\|^2 = \sum_{j=1}^{k} \frac{(\mathbf{u}_j^T \mathbf{b})^2}{\sigma_j^2}. \tag{8.10}$$

Similarly, the only difference with the formula for $\|\mathbf{x}_+\|^2$ is the upper limit on the sum.

Truncating the SVD thus decreases the norm of the solution:

$$\|\mathbf{x}_+\|^2 - \|\mathbf{x}_k\|^2 = \sum_{j=k+1}^{r} \frac{(\mathbf{u}_j^T \mathbf{b})^2}{\sigma_j^2}.$$

We conclude that

$$\|\mathbf{x}_k\| \textit{ increases} \text{ with } k, \text{ and } \|\mathbf{x}_k\| \leq \|\mathbf{x}_+\|.$$

By picking k so that σ_k is not too small, we can prevent $\|\mathbf{x}_k\|$ from being offensively large. But how well does $\mathbf{x}_k$ satisfy the original least squares problem we really want to solve? With the help of the SVD, we can readily check:

$$\begin{aligned}
\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|^2 &= \left\| \mathbf{b} - \Big(\sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \Big) \Big(\sum_{j=1}^{k} \frac{1}{\sigma_j} \mathbf{v}_j \mathbf{u}_j^T \mathbf{b} \Big) \right\|^2 \\
&= \left\| \mathbf{b} - \sum_{i=1}^{r} \sum_{j=1}^{k} \frac{\sigma_i}{\sigma_j} \mathbf{u}_i \mathbf{v}_i^T \mathbf{v}_j \mathbf{u}_j^T \mathbf{b} \right\|^2 \\
&= \left\| \mathbf{b} - \sum_{j=1}^{k} (\mathbf{u}_j^T \mathbf{b}) \mathbf{u}_j \right\|^2 \\
&= \left\| \sum_{j=k+1}^{m} (\mathbf{u}_j^T \mathbf{b}) \mathbf{u}_j \right\|^2 \\
&= \sum_{j=k+1}^{m} (\mathbf{u}_j^T \mathbf{b})^2.
\end{aligned}$$

Recall that, using $\widehat{\mathbf{U}} \in \mathbb{R}^{n \times n}$ from the full SVD,

$$\begin{aligned}
\mathbf{b} &= \widehat{\mathbf{U}} \widehat{\mathbf{U}}^T \mathbf{b} \\
&= \sum_{j=1}^{m} \mathbf{u}_j \mathbf{u}_j^T \mathbf{b} = \sum_{j=1}^{m} (\mathbf{u}_j^T \mathbf{b}) \mathbf{u}_j.
\end{aligned}$$

Compare this expression to the error from the least squares problem induced by the pseudoinverse solution, as characterized in equation (8.7). The residual $\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|$ is always at least as big as $\|\mathbf{b} - \mathbf{A}\mathbf{x}_+\|$, but perhaps it is not much larger if the omitted $\mathbf{u}_j^T \mathbf{b}$ coefficients are small:

Equation (8.7) for $\|\mathbf{b} - \mathbf{A}\mathbf{x}_+\|^2$ differs from the expression for $\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|^2$ only in the *lower limit* on the sum: for $\mathbf{x}_k$ we start at $j = k+1$; for $\mathbf{x}_+$ we started at $j = r+1$.

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|^2 - \|\mathbf{b} - \mathbf{A}\mathbf{x}_+\|^2 = \sum_{j=k+1}^{r} (\mathbf{u}_j^T \mathbf{b})^2.$$

We conclude that

$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|$ *decreases* with k, and $\|\mathbf{b} - \mathbf{A}\mathbf{x}_+\| \leq \|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|$.

In summary, using the truncated SVD can greatly reduce $\|\mathbf{x}_k\|$ in (8.6) by omitting the $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ terms for small σ_j, but the increase in the norm of the residual, $\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|$ is comparatively modest, only adding terms like $\mathbf{u}_j^T\mathbf{b}$. (See Figure 7.12 for a comparison of $\mathbf{u}_j^T\mathbf{b}$ and $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ for a practical example.)

Compare the pseudoinverse solution

$$\mathbf{x}_+ = \sum_{j=1}^{r} \frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j^T\mathbf{b} \tag{8.11}$$

to the truncated SVD approximation

$$\mathbf{x}_k = \sum_{j=1}^{k} \frac{1}{\sigma_j}\mathbf{v}_j\mathbf{u}_j^T\mathbf{b} \tag{8.12}$$

for the least squares problem $\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$:

- The norm of the truncated SVD *solution* is smaller:

$$\|\mathbf{x}_k\|^2 = \|\mathbf{x}_+\|^2 - \sum_{j=k+1}^{r} \frac{1}{\sigma_j^2}|\mathbf{u}_j^T\mathbf{b}|^2.$$

- The norm of the truncated SVD *residual* is larger:

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|^2 = \|\mathbf{b} - \mathbf{A}\mathbf{x}_+\|^2 + \sum_{j=k+1}^{r} |\mathbf{u}_j^T\mathbf{b}|^2.$$

In applications where $\sigma_{k+1}, \ldots, \sigma_r$ are small, the reduction in the norm of the solution can yield much more physically realistic answer by removing the artificial but overwhelming effects of noisy data, while the modest increase in the residual is not such a big concern.

8.1.1 Deblurring with truncated SVD regularization

Let us see how truncated SVD approximations perform in practice. In the process, we hope to get some instinct for how we should select the parameter k controlling the amount of truncation.

Again consider the blurring experiment explored in Section 7.5, using the same matrix $\mathbf{A}$ of dimension $n = 1000$ with the blurring kernel shown in Figure 7.4 with $z = 0.05$.

Recall the experiment shown in Figure 7.10. We took a known function $\mathbf{f}$, applied the blurring matrix $\mathbf{A}$ to get the "exact" image $\mathbf{b}$, then added noise to $\mathbf{b}$ to get $\mathbf{b}_{\text{noise}}$. We investigated how the noise

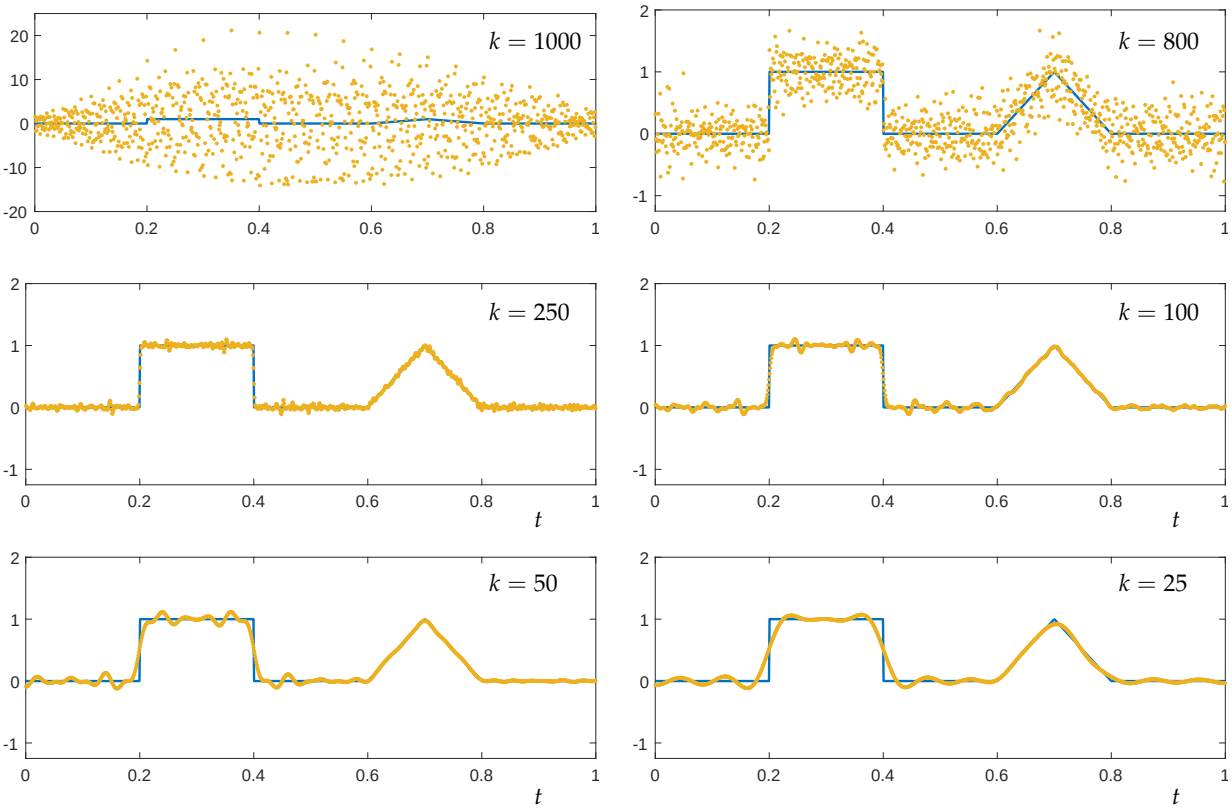


Figure 8.1: Recovered functions $\mathbf{f}_k$ using various values of k for the vector $\mathbf{b}_{\text{noise}}$ with noise level 10^{-4} used in Figure 7.10. For $k = 1000$ we have the "exact" solution, which is dominated by noise. SVD regularization gets a much better solution; among these plots, one might argue that $k = 250$ is "closest" to the desired solution.

level affected the accuracy of the recovered solution. We pick up the worst of these examples, where the Gaussian noise had standard deviation 10^{-4} and the recovered $\mathbf{f}$ was nonsense.

To simplify notation in the rest of this example, we will simply use "$\mathbf{b}$" to refer to the noisy vector $\mathbf{b}_{\text{noise}}$.

Let us try to recover $\mathbf{f}$ with better accuracy using the truncated SVD solution

$$\mathbf{f}_k = \sum_{j=1}^{k} \left(\frac{\mathbf{u}_j^T \mathbf{b}}{\sigma_j} \right) \mathbf{v}_j.$$

Figure 8.1 shows the results. You might expect that we would get the best result for k just a bit smaller than $n = 1000$. In fact, we get good results even for $k = 25$. Look back on the plot of the singular vectors in Figure 7.13. As j increases, the *frequency* (number of oscillations) increases. No surprise, then, that as k increases and we include terms involving $\mathbf{v}_j$ for larger values of j, we start to see finer and finer oscillations in the recovered $\mathbf{f}_k$. Of the examples plotted, perhaps $k = 250$ is the best. Notice that $k = 800$, which avoids the smallest 200 singular values, is already too large: the recovered solution is clearly polluted by noise.

Perhaps this is not a surprise, given the rapid decay of the initial singular values seen in Figure 7.11.

Picking the best k values is a nuanced judgement that could benefit from the insight of a domain scientist who is looking for a particular feature in a recovered image.

To help select an effective value of k, we draw the plot shown in

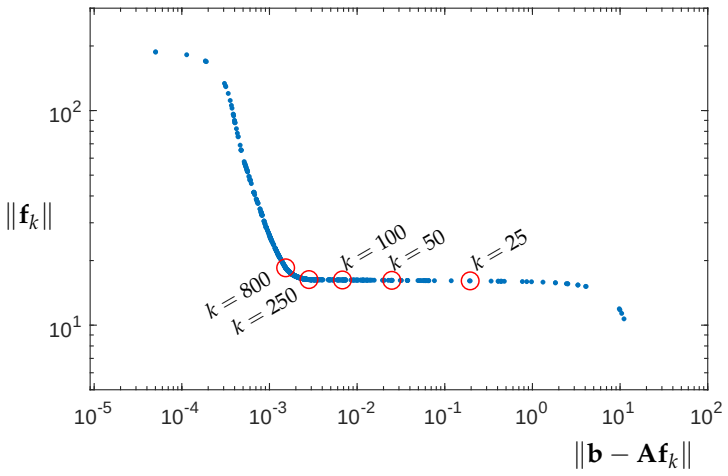


Figure 8.2: Plot illustrating the trade-off between minimizing $\|\mathbf{b} - \mathbf{A}\mathbf{f}_k\|$ and controlling the norm of the solution, $\|\mathbf{f}_k\|$, for $k = 1, \ldots, 999$. (For $k = 1000$, $\|\mathbf{b} - \mathbf{A}\mathbf{f}_k\| = 0$ but $\|\mathbf{f}_k\|$ is slightly larger than the largest value shown here.) The optimal value comes around the "corner" of the L-shape in this plot.

Figure 8.2. In this diagram the horizontal axis shows the norm of the residual, $\|\mathbf{b} - \mathbf{A}\mathbf{f}_k\|$, while the vertical axis shows the norm of the solution $\|\mathbf{f}_k\|$. (Because these quantities can differ over orders of magnitude, the axes are typically drawn with a logarithmic scale.) Notice that neither of these axes shows k: we plot $(\|\mathbf{b} - \mathbf{A}\mathbf{f}_k\|, \|\mathbf{f}_k\|)$ and must somehow indicate the k value associated with the resulting plot. However, since the residual norms *decrease monotonically with* k

$$\|\mathbf{b} - \mathbf{A}\mathbf{f}_1\| \leq \|\mathbf{b} - \mathbf{A}\mathbf{f}_2\| \leq \cdots \leq \|\mathbf{b} - \mathbf{A}\mathbf{f}_{n-1}\| \leq \|\mathbf{b} - \mathbf{A}\mathbf{f}_+\|$$

and the solution norms *increase monotonically with* k

$$\|\mathbf{f}_1\| \leq \|\mathbf{f}_2\| \leq \cdots \leq \|\mathbf{f}_{n-1}\| \leq \|\mathbf{f}_+\|,$$

the plot must move from the bottom-right to the top-left as k increases. We highlight the five values of k used in Figure 8.1. (Since $\|\mathbf{b} - \mathbf{A}\mathbf{f}_+\| = 0$ in this case, this point would fall off the axes shown.)

Notice that this plot has a prominent "L" shape: take k too small, and the residual norm is quite large (bottom right). As we gradually increase k we tend to make great improvements to $\|\mathbf{b} - \mathbf{A}\mathbf{f}_k\|$ while making small increases in $\|\mathbf{x}_k\|$. Eventually, however, this trend changes: we reach a stage where increasing k only decreases $\|\mathbf{b} - \mathbf{A}\mathbf{f}_k\|$ slightly, while it significantly adds to $\|\mathbf{f}_k\|$. In this regime, we expect that we are adding those large $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ coefficients that are heavily influenced by the noise. We thus want to pick a k that occurs somewhere near the bend in the "L" shape. (Notice that many values of k fall between $k = 250$ and $k = 800$ in the plot.) Selecting k is something of an art, but one that can pay off with excellent solutions seen in Figure 8.1.

8.1.2 Computing the SVD regularized solution

```
U,S,Vt = la.svd(A)
xk = (Vt[0:k,:].T)*(1/(S[0:k]))@(U[:,0:k].T@b)
```

8.2 Tikhonov Regularization (Ridge Regression)

The truncated SVD has great appeal: one must appreciate the simplicity and potency of this approach, which seemed quite effective for the deblurring example in Figure 8.1. However, to implement this method we must first compute the singular value decomposition of $\mathbf{A}$, which can be a costly step when $\mathbf{A}$ is a large matrix. Often such large matrices are "sparse," meaning most entries are zero; when m and n are both large, it could be quite an obstacle to compute even the reduced form of the SVD of $\mathbf{A}$. Another approach to taming small singular values is equally intuitive but more computationally appealing: *Tikhonov regularization*.

The highlighted paragraph on page 122 describes conflicting tensions that arise when solving ill-posed problems: we seek to make the least-squares residual $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ as small as possible, while also controlling the size of the solution $\|\mathbf{x}\|$. The fundamental problem is that the $\mathbf{x}$ that minimizes $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ often gives large $\|\mathbf{x}\|$. We are willing to accept a *suboptimal* $\mathbf{x}$ that gives a *slightly* larger $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ but a *much smaller* $\|\mathbf{x}\|$.

Why not combine these two goals into one optimization problem? Replace the usual least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$$

with the *penalized* problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda^2\|\mathbf{x}\|^2 \tag{8.13}$$

for some choice of the *regularization parameter* λ. Two fundamental questions arise.

- How should one choose λ?
- Given λ, how does one find the optimal $\mathbf{x}$ in (8.13)?

We shall tackle these questions in reverse order.

8.2.1 Solving regularized least squares problems

Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$. Define the *augemented* matrix and vector

$$\mathbf{A}_\lambda = \begin{bmatrix} \mathbf{A} \\ \lambda\mathbf{I} \end{bmatrix} \in \mathbb{R}^{(m+n)\times n}, \qquad \widehat{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{m+n}.$$

Note that the matrix $\lambda\mathbf{I}$ has size $n \times n$, while the zero vector in $\widehat{\mathbf{b}}$ is of length n.

and consider the alternative least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x}\|. \tag{8.14}$$

Perhaps it helps to look at the structure of $\mathbf{A}_\lambda$ and $\widehat{\mathbf{b}}$ for a small example. The general $m = 3$ and $n = 2$ case has the structure

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix},$$

and so the augmented matrix and vector become

$$\mathbf{A}_\lambda = \begin{bmatrix} \mathbf{A} \\ \lambda\mathbf{I} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ \lambda & 0 \\ 0 & \lambda \end{bmatrix}, \qquad \widehat{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ 0 \\ 0 \end{bmatrix}. \tag{8.15}$$

We can solve the least squares problem (8.14) using the conventional technology we studied in Chapter 7. In particular, we could simply use the pseudoinverse to write down the solution

$$\mathbf{x}_\lambda = \mathbf{A}_\lambda^+ \widehat{\mathbf{b}}.$$

Before exploring this solution, we want to understand how the least squares problem (8.14) relates to the penalized problem (8.13) that we really want to solve. First note that for any $\mathbf{x} \in \mathbb{R}^n$,

$$\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{A} \\ \lambda\mathbf{I} \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{A}\mathbf{x} \\ \lambda\mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} - \mathbf{A}\mathbf{x} \\ -\lambda\mathbf{x} \end{bmatrix}.$$

Square the norm of this residual and expand:

$$\begin{aligned} \|\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x}\|^2 &= (\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x})^T (\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x}) \\ &= \begin{bmatrix} \mathbf{b} - \mathbf{A}\mathbf{x} \\ -\lambda\mathbf{x} \end{bmatrix}^T \begin{bmatrix} \mathbf{b} - \mathbf{A}\mathbf{x} \\ -\lambda\mathbf{x} \end{bmatrix} \\ &= \left[\begin{array}{cc} (\mathbf{b} - \mathbf{A}\mathbf{x})^T & (-\lambda\mathbf{x})^T \end{array} \right] \begin{bmatrix} \mathbf{b} - \mathbf{A}\mathbf{x} \\ -\lambda\mathbf{x} \end{bmatrix} \Bigg) \\ &= (\mathbf{b} - \mathbf{A}\mathbf{x})^T (\mathbf{b} - \mathbf{A}\mathbf{x}) + (-\lambda\mathbf{x})^T (-\lambda\mathbf{x}) \\ &= \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda^2 \|\mathbf{x}\|^2. \end{aligned}$$

We have just shown that

$$\|\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x}\|^2 = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda^2 \|\mathbf{x}\|^2,$$

and so

the $\mathbf{x} \in \mathbb{R}^n$ that solves the least squares problem (8.14) also solves the penalized least squares problem (8.13).

Take a few moments to think about $\mathbf{A}_\lambda$. To understand if (8.14) has a unique solution, we need to understand the rank and null space of $\mathbf{A}_\lambda$. What is $\text{rank}(\mathbf{A}_\lambda)$?

You can determine the rank in several ways; we will provide a detailed discussion, as a review of the fundamental subspaces. (You could jump to this conclusion more rapidly if you like.) It might help to look back on the augmented matrix for the $m = 3$ and $n = 2$ case in (8.15) for guidance.

Regardless of $\mathbf{A}$, the last n rows of the augmented matrix

$$\mathbf{A}_\lambda = \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix},$$

which are just $\lambda \mathbf{I} \in \mathbb{R}^{n \times n}$ with $\lambda > 0$, must be linearly independent. We conclude that the dimension of the row space must satisfy

$$\dim(\mathcal{R}(\mathbf{A}_\lambda^T)) \geq n.$$

However, since $\mathbf{A}_\lambda$ only has n columns, we must have

$$\dim(\mathcal{R}(\mathbf{A}_\lambda)) \leq n.$$

Since

$$\dim(\mathcal{R}(\mathbf{A}_\lambda^T)) = \dim(\mathcal{R}(\mathbf{A}_\lambda)) = \text{rank}(\mathbf{A}),$$

we conclude that

$$\text{rank}(\mathbf{A}_\lambda) = n,$$

and so $\mathbf{A}_\lambda$ must have n linearly independent columns.

Now as a consequence of the Fundamental Theorem of Linear Algebra (Theorem 5.5) or the rank-nullity theorem,

$$\begin{aligned} \dim(\mathcal{N}(\mathbf{A}_\lambda)) &= n - \dim(\mathcal{R}(\mathbf{A}_\lambda^T)) \\ &= n - \text{rank}(\mathbf{A}_\lambda), \end{aligned}$$

and so $\dim(\mathcal{N}(\mathbf{A}_\lambda)) = 0$, and hence $\mathbf{A}_\lambda$ must have a trivial null space,

$$\mathcal{N}(\mathbf{A})_\lambda) = \{\mathbf{0}\}.$$

Thus, referring to the flowchart in Figure 7.2, we conclude that the least squares problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x}\|.$$

has a unique solution. Indeed, we can find that solution by solving the *normal equations*

$$\mathbf{A}_\lambda^T \mathbf{A}_\lambda \mathbf{x}_\lambda = \mathbf{A}_\lambda^T \widehat{\mathbf{b}}.$$

You might also study the rank of $\mathbf{A}_\lambda$ by computing its singular values. Recall that the singular values are square roots of the eigenvalues of $\mathbf{A}_\lambda^T \mathbf{A}_\lambda$. Since

$$\mathbf{A}_\lambda^T \mathbf{A}_\lambda = \mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{I},$$

you can see that the eigenvalues of $\mathbf{A}_\lambda^T \mathbf{A}_\lambda$ are just the eigenvalues of $\mathbf{A}^T \mathbf{A}$, plus λ^2: for if

$$\mathbf{A}^T \mathbf{A} \mathbf{v}_j = \sigma_j^2 \mathbf{v}_j,$$

then

$$\mathbf{A}_\lambda^T \mathbf{A}_\lambda \mathbf{v}_j = (\mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{I}) \mathbf{v}_j = (\sigma_j^2 + \lambda^2) \mathbf{v}_j.$$

Thus we see that for any nonzero λ,

$$j\text{th singular value of } \mathbf{A}_\lambda = \sqrt{\sigma_j + \lambda^2} > 0,$$

even if $\sigma_j = 0$. Since $\mathbf{A}_\lambda$ has n nonzero singular values, its rank must be n. The right singular vectors $\mathbf{v}_j$ of $\mathbf{A}_\lambda$ are also the right singular vectors of $\mathbf{A}$.

The left-hand side is

$$\mathbf{A}_\lambda^T \mathbf{A}_\lambda = \begin{bmatrix} \mathbf{A}^T & \lambda \mathbf{I}^T \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix} = \mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{I},$$

while the right-hand side has the simple form

$$\mathbf{A}_\lambda^T \widehat{\mathbf{b}} = \begin{bmatrix} \mathbf{A}^T & (-\lambda \mathbf{I})^T \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} = \mathbf{A}^T \mathbf{b} \in \mathbb{R}^n.$$

Let us summarize where we now stand.

For any $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\lambda \neq 0$, define

$$\mathbf{A}_\lambda = \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix}, \qquad \widehat{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}.$$

The *augmented matrix* $\mathbf{A}_\lambda$ has

$$\operatorname{rank}(\mathbf{A}_\lambda) = n$$

and hence $\mathcal{N}(\mathbf{A}_\lambda) = \{\mathbf{0}\}$. Thus the least squares problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\widehat{\mathbf{b}} - \mathbf{A}_\lambda \mathbf{x}\|.$$

has the unique solution

$$\begin{aligned} \mathbf{x}_\lambda &= (\mathbf{A}_\lambda^T \mathbf{A}_\lambda)^{-1} \mathbf{A}_\lambda^T \widehat{\mathbf{b}} \\ &= (\mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{I})^{-1} \mathbf{A}^T \mathbf{b}, \end{aligned}$$

which is also the unique solution to the penalized problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda \|\mathbf{x}\|^2.$$

Let us example the formula for the exact solution

$$\mathbf{x}_\lambda = (\mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{I})^{-1} \mathbf{A}^T \mathbf{b}.$$

We will work with the SVD of $\mathbf{A}$, expressed in the dyadic form

$$\mathbf{A} = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{8.16}$$

Equivalently, we could express $\mathbf{x}_\lambda = \mathbf{A}_\lambda^+ \widehat{\mathbf{b}}$ and work out a formula for $\mathbf{A}_\lambda^+$.

For convenience, we write this sum up to n, instead of the usual $r = \operatorname{rank}(\mathbf{A})$. In the case that $\operatorname{rank}(\mathbf{A}) < n$: for $j = r+1, \ldots, n$,

- define $\sigma_j = 0$;
- let $\mathbf{u}_j$ be a unit vector orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}$;
- let $\mathbf{v}_j$ be a unit vector orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_{j-1}$.

With this convention, the form (8.16) holds for any $\mathbf{A} \in \mathbb{R}^{m\times n}$. We can then write

$$\mathbf{A}^T\mathbf{A} + \lambda^2\mathbf{I} = \sum_{j=1}^{n} \sigma_j^2 \mathbf{v}_j\mathbf{v}_j^T + \lambda^2 \sum_{j=1}^{n} \mathbf{v}_j\mathbf{v}_j^T = \sum_{j=1}^{n} (\sigma_j^2 + \lambda^2)\mathbf{v}_j\mathbf{v}_j^T,$$

which can be readily inverted:

$$(\mathbf{A}^T\mathbf{A} + \lambda^2\mathbf{I})^{-1} = \sum_{j=1}^{n} \frac{1}{\sigma_j^2 + \lambda^2}\mathbf{v}_j\mathbf{v}_j^T.$$

From this we can compute

$$\begin{aligned}
\mathbf{x}_\lambda &= (\mathbf{A}^T\mathbf{A} + \lambda^2\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b} \\
&= \left(\sum_{j=1}^{n} \frac{1}{\sigma_j^2 + \lambda^2}\mathbf{v}_j\mathbf{v}_j^T \right) \left(\sum_{\ell=1}^{r} \sigma_\ell \mathbf{v}_\ell \mathbf{u}_\ell^T \right) \mathbf{b} \\
&= \sum_{j=1}^{r} \frac{\sigma_j}{\sigma_j^2 + \lambda^2} (\mathbf{u}_j^T\mathbf{b})\, \mathbf{v}_j,
\end{aligned}$$

using orthogonality of the singular vectors, as usual. We summarize:

The unique solution to the regularized least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda^2\|\mathbf{x}\|^2$$

is given by

$$\mathbf{x}_\lambda = \sum_{j=1}^{r} \frac{\sigma_j}{\sigma_j^2 + \lambda^2} (\mathbf{u}_j^T\mathbf{b})\mathbf{v}_j. \tag{8.17}$$

Recall that for $\mathbf{V} \in \mathbb{R}^{n\times n}$ with $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, we can write

$$\mathbf{I} = \mathbf{V}\mathbf{V}^T = \sum_{j=1}^{n} \mathbf{v}_j\mathbf{v}_j^T.$$

8.2.2 *Filtering singular values*

Contrast these three formulas:

pseudoinverse solution: $\mathbf{x}_+ = \sum_{j=1}^{r} \frac{1}{\sigma_j} (\mathbf{u}_j^T\mathbf{b})\, \mathbf{v}_j$

truncated SVD solution: $\mathbf{x}_k = \sum_{j=1}^{k} \frac{1}{\sigma_j} (\mathbf{u}_j^T\mathbf{b})\, \mathbf{v}_j$

Tikhonov solution: $\mathbf{x}_\lambda = \sum_{j=1}^{r} \frac{\sigma_j}{\sigma_j^2 + \lambda^2} (\mathbf{u}_j^T\mathbf{b})\mathbf{v}_j$

All three of these "solutions" involve the terms $(\mathbf{u}_j^T\mathbf{b})\mathbf{v}_j$, *but in each case these terms are scaled by a different function of the singular value* σ_j.

Indeed, these functions of the singular values distinguish the three solutions. We can absorb them in the general form

$$\mathbf{x} = \sum_{j=1}^{r} \phi(\sigma_j)\,(\mathbf{u}_j^T \mathbf{b})\,\mathbf{v}_j,$$

for different *filter functions* $\phi(\cdot)$.

We can characterize these filter functions as follows:

$$\textit{pseudoinverse filter}: \qquad \phi(\sigma) = \frac{1}{\sigma}.$$

$$\textit{truncated SVD filter}: \quad \phi(\sigma) = \begin{cases} \dfrac{1}{\sigma}, & \sigma \geq \sigma_k; \\ 0, & \sigma < \sigma_k. \end{cases}$$

$$\textit{Tikhonov filter}: \qquad \phi(\sigma) = \frac{\sigma}{\sigma^2 + \lambda^2}.$$

Figure 8.3 illustrates these three filter functions (using SVD truncation for $\sigma_k = 10^{-2}$ and Tikhonov parameter $\lambda = 10^{-2}$).

Let us collect some helpful properties about the Tikhonov filter.

- *The Tikhonov filter is always smaller than the pseudoinverse filter.* For all $\lambda > 0$,
$$\frac{\sigma}{\sigma^2 + \lambda^2} < \frac{1}{\sigma^2}.$$

- *The Tikonov filter attains maximum value* $1/(2\lambda)$ *at* $\sigma = \lambda$. For the Tikhonov filter, notice that
$$\phi'(\sigma) = \frac{1}{\sigma^2 + \lambda^2} - \frac{2\sigma^2}{(\sigma^2 + \lambda^2)^2},$$
and so solving $\phi'(\sigma) = 0$ shows that the filter function attains its maximum when
$$\sigma = \lambda, \qquad f(\sigma) = \frac{1}{2\lambda}.$$

- *The Tikhonov filter converges to zero in the extremes* $\sigma \to 0$ *and* $\sigma \to \infty$. This property is easy to see from the formula
$$\phi(\sigma) = \frac{\sigma}{\sigma^2 + \lambda^2}.$$
Notice that ϕ's behavior as $\sigma \to \infty$ is consistent with the filter functions for the pseudoinverse and truncated SVD. However, the Tikhonov filter's behavior as $\sigma \to 0$ (i.e., the way it handles small singular values) is distinct: unlike the pseudoinverse filter, it does not blow up; unlike the SVD truncation, it always allows even the small singular values (and the corresponding right singular vector $\mathbf{v}_j$) to exert some influence.

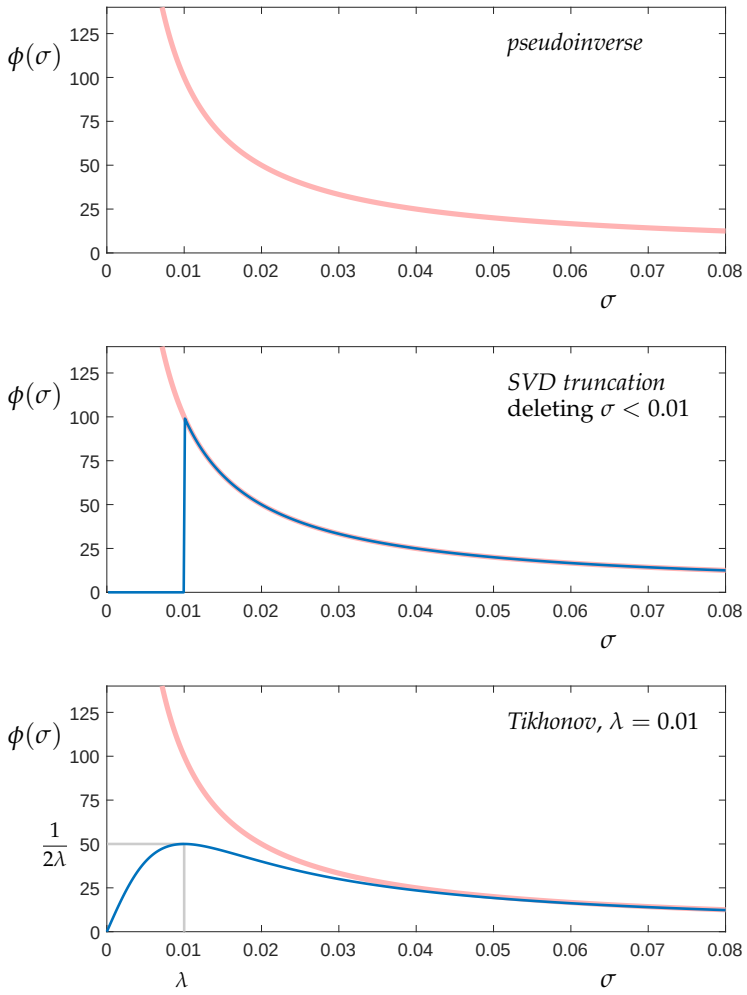


Figure 8.3: Filter functions for the standard pseudoinverse ($\phi(\sigma) = 1/\sigma$, top, and superimposed in red on the other plots), the truncated SVD regularization (middle, deleting all singular values below $\sigma \leq 0.01$), and the Tikhonov regularization (bottom, $\phi(\sigma) = \sigma/(\sigma^2 + \lambda^2)$ for $\lambda = 0.01$).

- *As λ increases, the Tikhonov filter gets smaller.*
  For $\lambda_1 > \lambda_2$,
  $$\frac{\sigma}{\sigma^2 + \lambda_1^2} < \frac{\sigma}{\sigma^2 + \lambda_2^2},$$
  since the denominator is smaller on the right-hand side. Indeed, as $\lambda \to \infty$ with fixed $\sigma > 0$, $\phi(\sigma) \to 0$. This property suggests that by taking λ too large, we will excessively suppress the size of $\mathbf{x}_\lambda$ that solves from the least squares problem
  $$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda \|\mathbf{x}\|^2.$$
  As $\lambda \to \infty$, the $\lambda\|\mathbf{x}\|^2$ term will dominate, driving $\mathbf{x}_\lambda \to \mathbf{0}$. (We will see a hint of this behavior in Figure 8.4.)
  As $\lambda \to 0$, the $\lambda^2\|\mathbf{x}\|^2$ term exerts little influence on the least squares problem, and $\mathbf{x}_\lambda$ will be quite close to the pseudoinverse solution $\mathbf{x}_+$.

8.2.3 *Selecting the regularization parameter*

How then should one select the regularization parameter λ to yield the best results? One seeks to strike a perfect balance between keeping $\|\mathbf{x}_\lambda\|$ at a moderate size while making $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ as small as possible. One way to select λ is to create a plot with $\log \|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ on the horizontal axis, and $\log \|\mathbf{x}_\lambda\|$ on the vertical axis, sampled over a wide range of λ values (varying over orders of magnitude). Often this plot shows a distinct bend, as we will see momentarily in Figure 8.5. For many applications, the best choice for λ will yield values of $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ and $\|\mathbf{x}_\lambda\|$ that land at the sharp bend in this "L curve."

For many more details of regularization problem, ranging from applications to algorithms, we recommend the excellent introductory book: P. C. Hansen, *Discrete Inverse Problems: Insight and Algorithms*, SIAM, Philadelphia, 2010.

8.2.4 *Deblurring with Tikhonov regularization*

Figure 8.4 applies Tikhonov regularization to our running example of one-dimensional image deblurring with a blurring matrix $\mathbf{A}$ of dimension $n = 1000$ generated from the hat-function kernel (7.4) with $z = 0.05$. The right-hand side $\mathbf{b}$ was generated from adding Gaussian noise (standard deviation 10^{-4}) to the exact right-hand side $\mathbf{A}^{-1}\mathbf{f}$ for known $\mathbf{f}$. The yellow dots show how well Tikhonov regularization recovers the exact $\mathbf{f}$ from the noisy right-hand side.

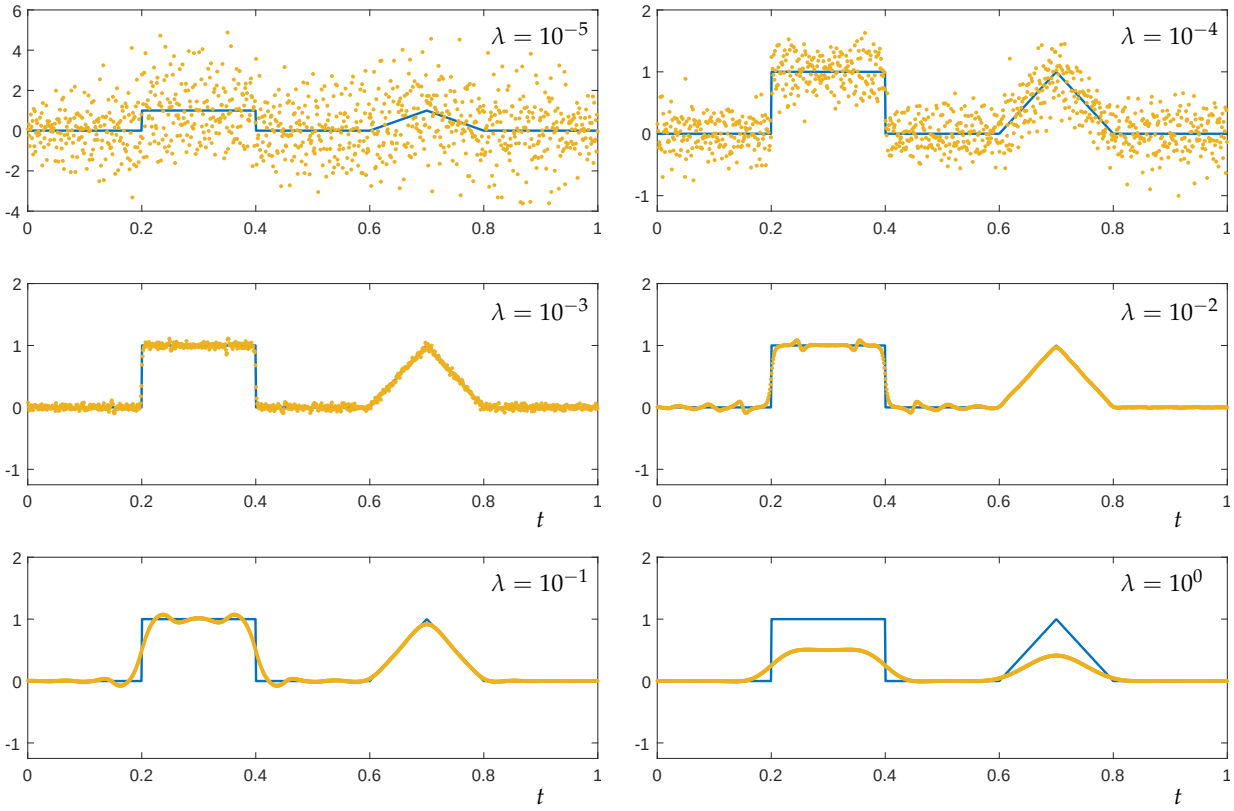


Figure 8.4: Recovered functions $\mathbf{f}_\lambda$ using various values of λ, for the vector $\mathbf{b}_{\text{noise}}$ with noise level 10^{-4} used in Figure 7.10.

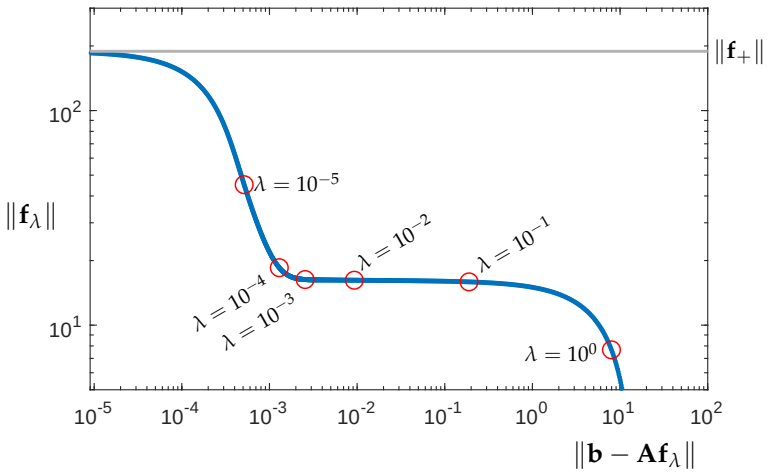


Figure 8.5: The trade-off between minimizing $\|\mathbf{b} - \mathbf{A}\mathbf{f}_\lambda\|$ and controlling the norm of the solution, $\|\mathbf{f}_\lambda\|$: as λ increases, so does the penalty on large $\|\mathbf{f}_\lambda\|$ values; at the cost of increasing the residual norm $\|\mathbf{b} - \mathbf{A}\mathbf{f}_\lambda\|$. (Since this problem has an exact solution $\mathbf{f}_+ = \mathbf{A}^{-1}\mathbf{b}$, $\|\mathbf{f}_+\|$ forms an upper bound on $\|\mathbf{f}_\lambda\|$. The "optimal" value of λ falls around the corner of this "L curve".

Study these plots to develop insight about the influence of the regularization parameter λ on the recovered solutions.

- When λ is too small (10^{-5} and 10^{-4}), the answer remains polluted with noise, as in the case of the corresponding pseudoinverse solution (bottom-right plot in Figure 7.10).
- When $\lambda = 10^{-3}$ and $\lambda = 10^{-2}$, the solutions quite nice. Notice how well the 10^{-2} captures the triangular shape, in particular. For $\lambda = 10^{-1}$, the resolution of the edge of the rectangular portion on the left degrades.
- When λ is too large (10^{0} here), the solution is excessively suppressed by the $\lambda^2\|\mathbf{f}\|^2$ penalty term; the result is a poor approximation of the true solution.

Figure 8.5 shows that the choice of λ can be a challenge. The corner of this "L curve" comes somewhere between $\lambda = 10^{-4}$ and $\lambda = 10^{-3}$. Figure 8.4 suggests that $\lambda = 10^{-4}$ is still too small: the recovered solution still exhibits considerable noise. The solution looks better with 10^{-3}, but still shows consider chatter. Of the solutions plotted in Figure 8.4, one might prefer $\lambda = 10^{-2}$ as the "best approximation", even though the corresponding $(\|\mathbf{b} - \mathbf{A}\mathbf{f}_\lambda\|, \|\mathbf{f}_\lambda)$ point falls a bit after the corner in the "L curve".

8.2.5 *Selecting the regularization parameter*

For many ill-posed problems, like we observed in Figure 8.5, the plot of $\|\mathbf{x}_\lambda\|$ versus $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ takes a distinctive shape resembling the letter L, and hence such plots are called *L curves*.

Start at the top of this L. As you descend the vertical, $\|\mathbf{x}_\lambda\|$ decreases while $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ does not grow too much: a small compro-

mise in the size of the misfit $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ amounts to a significant reduction in the norm of the solution $\|\mathbf{x}_\lambda\|$, allowing us to bring that large large value under control. Now as you turn the corner, the opposite occurs: $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|$ increases rapidly, with little additional reduction in $\|\mathbf{x}_\lambda\|$: we gain little more control of the norm of the solution while significantly increasing the misfit.

Thus, we seek λ that gives $(\|\mathbf{b} - \mathbf{A}\mathbf{x}_\lambda\|, \|\mathbf{x}_\lambda\|)$ near the corner of the L. Here is the tricky part: ideally we would find this appealing λ value *without drawing the L curve*, since drawing this plot requires us to guess a suitable range $[\lambda_{\min}, \lambda_{\max}]$, fill this interval with many λ values, and then minimize $\|\mathbf{A}_\lambda\mathbf{x} - \widehat{\mathbf{b}}\|$ for each of these λ. This computation becomes quite expensive; it can be expedited by computing the SVD of $\mathbf{A}$ and using the formula

To draw the L curve, we would typically space these λ logarithmically between $\lambda_{\min}$ and $\lambda_{\max}$, since the interval typically spans several orders of magnitude, e.g., $[\lambda_{\min}, \lambda_{\max}] = [10^{-6}, 10^{0}]$.

$$\mathbf{x}_\lambda = \sum_{j=1}^{r} \frac{\sigma_j}{\sigma_j^2 + \lambda^2} (\mathbf{u}_j^T \mathbf{b}) \mathbf{v}_j$$

to form each solution, rather than solving a least squares problem for each value of λ.

Numerous techniques have been proposed for finding the bend in the L curve more efficiently. Look back to Figure 8.5. You can view this L curve as a function, a function that is always decreasing and hence has a negative derivative. The bend in the curve occurs where there is an abrupt change in the derivative from quite large in magnitude to being quite small. Note one key subtlety: this "function" is not a simple function with λ running on the horizontal axis, but it is a function defined by plotting one function of λ ($\|\mathbf{x}_\lambda\|$) against another ($\|\mathbf{A}\mathbf{x}_\lambda - \mathbf{b}\|$). Approximating this point calls for some careful implicit differentiation.

For details, see P. C. Hansen, *Discrete Inverse Problems: Insight and Algorithms*, SIAM, Philadelphia, 2010.

We will instead discuss another method, one that requires more intensive computations but gives some statistical motivation for the choice of λ. The approach is based on *cross-validation*. This essence of this approach, initiated by Golub, Heath, and Wahba (1979), is:

> Pick the regularization parameter λ to be the value that minimizes the influence of any single row $\mathbf{b} - \mathbf{A}\mathbf{x}$ on the approximation $\mathbf{x}_\lambda$.

We shall sketch out only the simplest possible method in this vein. Start by writing $\mathbf{A}$ by *rows*, with $\mathbf{a}_j^T \in \mathbb{R}^{1\times n}$ denoting the jth row, so $\mathbf{b} - \mathbf{A}\mathbf{x}$ has the form

$$\mathbf{b} - \mathbf{A}\mathbf{x} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} - \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_m^T \end{bmatrix} \mathbf{x}.$$

Then the jth row of $\mathbf{b} - \mathbf{A}\mathbf{x}$ is then $b_j - \mathbf{a}_j^T\mathbf{x}$.

For each value of λ, we will define a cross-validation score $C(\lambda)$. By this measure the "optimal" value of λ will be the one that minimizes $C(\lambda)$. Here is a sketch of the computation of $C(\lambda)$.

Let $\mathbf{A}^{(j)} \in \mathbb{R}^{(m-1)\times n}$ and $\mathbf{b}^{(j)} \in \mathbb{R}^{m-1}$ denote $\mathbf{A}$ and $\mathbf{b}$ with the jth row and entry removed.

For every λ value:

- For each row $j = 1, \ldots, m$ of $\mathbf{A}$:
  - Solve the regularized least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b}^{(j)} - \mathbf{A}^{(j)}\mathbf{x}\|^2 + \lambda^2\|\mathbf{x}\|^2$$

    with the jth row omitted; call the minimizer $\mathbf{x}_\lambda^{(j)}$.
  - Compute how well the approximation $\mathbf{x}_\lambda^{(j)}$ satisfies jth equation (which was not used to find the approximation $\mathbf{x}_\lambda^{(j)}$):

$$r_{j,\lambda} := b_j - \mathbf{a}_j^T\mathbf{x}_\lambda^{(j)}.$$

- Then the cross-validation score for λ is the average of the squares of the errors in the jth equations:

$$C(\lambda) := \frac{r_{1,\lambda}^2 + r_{2,\lambda}^2 + \cdots + r_{m,\lambda}^2}{m}.$$

No doubt this algorithm appears extremely expensive: we are solving m regularization problems of size $(m-1)\times n$ for every value of λ! Thankfully, all of these computations can be accelerated by computing a single SVD of $\mathbf{A}$ and using some slick linear algebra, the details of which are too much to dig into here. If one knows that the minimal λ falls in some bracket, say $[\lambda_{\min}, \lambda_{\max}]$, then one can minimize $C(\lambda)$, using, for example, the trisection algorithm.

For details, see section 12.1.3 of G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins, Baltimore, 1996.

Figure 8.6 shows the function $C(\lambda)$ for the deblurring example. This function attains a minimum value of $\lambda_{\text{cv}} \approx 1.206 \times 10^{-3}$. The solution $\mathbf{f}_{\lambda_{\text{cv}}}$ for this value puts $(\|\mathbf{b} - \mathbf{A}\mathbf{f}_{\lambda_{\text{cv}}}\|, \|\mathbf{f}_{\lambda_{\text{cv}}}\|)$ just after the bend in the L curve, which agrees with the results shown in Figure 8.4. The bottom plot in Figure 8.6 shows that λ_{cv} does indeed deliver a nice solution, one that looks good to the eye but also has added statistical justification as the minimizer of the cross-validation objective function.

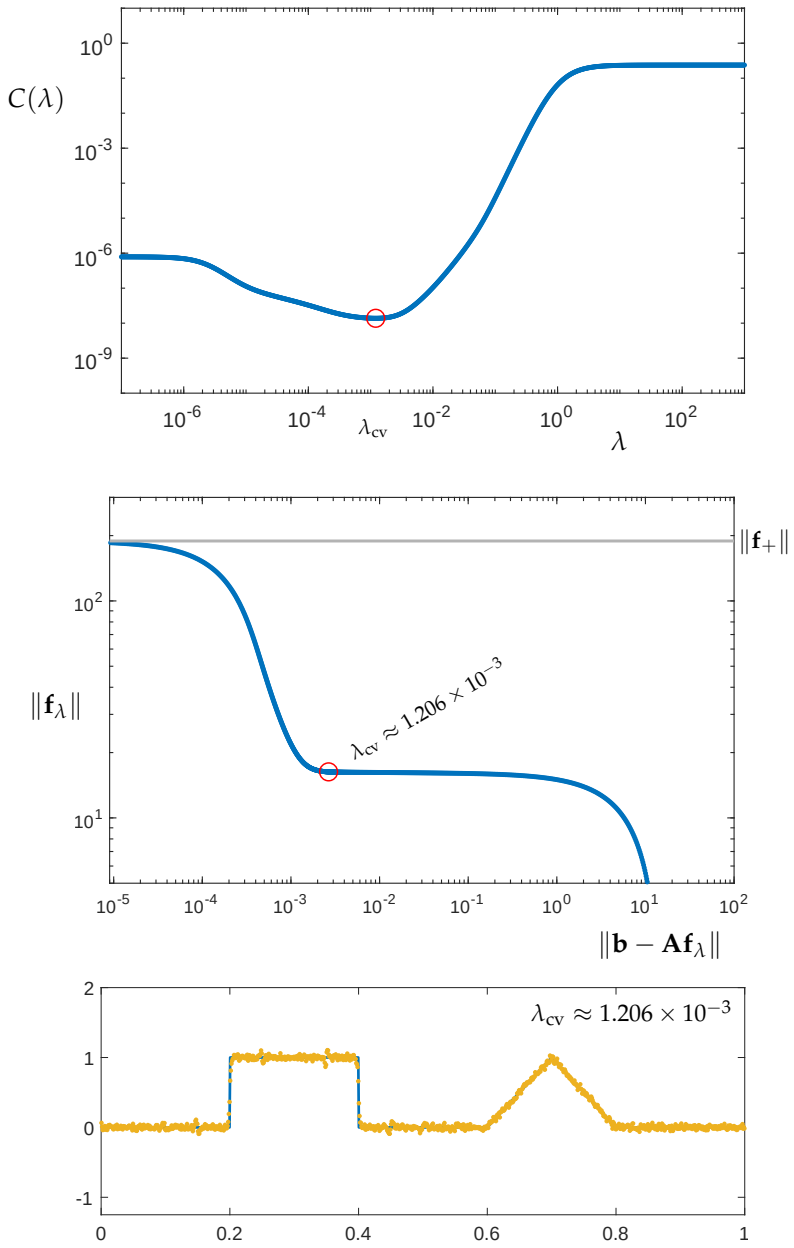


Figure 8.6: The cross-validation objective function $C(\lambda)$ (top), attaining its minimum at $\lambda_{\text{cv}} \approx 1.206 \times 10^{-3}$ for the deblurring example. This value λ_{cv} gives $(\|\mathbf{b} - \mathbf{A}\mathbf{f}_\lambda\|, \|\mathbf{f}_\lambda\|)$ just after the bend in the L curve (middle). The resulting solution $\mathbf{f}_{\lambda_{\text{cv}}}$ gives a good approximation to the true solution (bottom).

8.2.6 Computational complexity

We can solve the penalized least squares problem without computing a singular value decomposition. If one of the problems in (**??**) has a unique solution, the other does as well. We shall compute that solution $\mathbf{x}_\lambda$ using the conventional least squares problem (8.14).

Curiously, the default least squares solver in Python's SciPy library, `linalg.lstsq`, solves generic least squares problems by com-

puting the SVD and effectively using pseudoinverse solution, complete with a threshold for truncating small singular values. While this algorithm benefits from favorable numerical properties, for many problems methods based on the $\mathbf{A} = \mathbf{QR}$ factorization will be a bit faster.

8.2.7 History

The idea of "ridge regression" was introduced by Arthur Hoerl (who was working at DuPont at the time) in the chemical engineering literature in the 1960s, and then further investigated in a 1970 article by Hoerl and Kennard in *Technometrics*.

Roger Hoerl writes that his father was motivated by "the frequent occurrence of nonsensical estimates from least squares multiple regression", which is precisely the reason we have been exploring these ideas. (See R. Hoerl, "Ridge Analysis 25 Years Later," *American Statistician* 39 (1985) 186–192.)

8.3 Blurring and deblurring in two dimensions

This section is a companion piece to Section 7.5, extending the mathematical framework for image deblurring into two dimensions and hence applying to our usual notion of an "image."

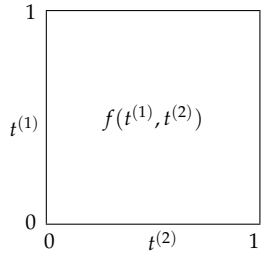
Digital images are matrices, with each entry corresponding to the color value of a single pixel. We shall only consider a simple set-up, square images with only one color intensity. As you read along, consider how these ideas could be extended into rectangular images and three color matrices for red, green, and blue.

As in the one-dimensional case, we base our model for blurring on "images" that are functions of continuous variables. In particular, we consider the true image to be a function

$$f(t^{(1)}, t^{(2)})$$

of two variables that range over the unit interval:

$$0 \leq t^{(1)}, t^{(2)} \leq 1.$$



The real number $f(t^{(1)}, t^{(2)})$ describes the color intensity at the point $(t^{(1)}, t^{(2)})$ in the unit square. The first parameter $t^{(1)}$ gives the *vertical* location, while the second parameter $t^{(2)}$ gives the *horizontal* location.

When we acquire the image through a camera, the device effectively blurs the true image, resulting in the blurred function

$$b(s^{(1)}, s^{(2)}),$$

also evaluated on the unit square, $0 \leq s^{(1)}, s^{(2)} \leq 1$. How do we model the transformation of the "true object" f into the blurred version b?

Generalize the approach we used in one dimension, where the blurred value $b(s)$ is obtained by integrating f against the blurring kernel $h(s,t)$ over the entire domain $t \in [0,1]$:

$$b(s) = \int_0^1 h(s,t) f(t)\, \mathrm{d}t.$$

Follow the same principle in two dimensions. Now the integral must traverse both $0 \le t^{(1)} \le 1$ and $0 \le t^{(2)} \le 1$ to cover the unit square:

$$b(s^{(1)}, s^{(2)}) = \int_0^1 \int_0^1 h\left(\begin{bmatrix} s^{(1)} \\ s^{(2)} \end{bmatrix}, \begin{bmatrix} t^{(1)} \\ t^{(2)} \end{bmatrix} \right) f(t^{(1)}, t^{(2)})\, \mathrm{d}t^{(1)}\, \mathrm{d}t^{(2)}. \quad (8.18)$$

Notice now that the *blurring kernel* h is a function of four variables. We can consolidate the two pairs of arguments into the vectors

$$\mathbf{s} = \begin{bmatrix} s^{(1)} \\ s^{(2)} \end{bmatrix}, \qquad \mathbf{t} = \begin{bmatrix} t^{(1)} \\ t^{(2)} \end{bmatrix}.$$

Then we could define the *Gaussian kernel*

$$h(\mathbf{s}, \mathbf{t}) = \frac{1}{2\pi z^2} \exp\left(\frac{-\|\mathbf{s} - \mathbf{t}\|^2}{2z^2} \right) \quad (8.19)$$

where z is a parameter that controls the severity of blurring: the larger z, the greater the influence remote values of f have on b, and hence the stronger the blurring effect. Our examples in these notes will use a kernel that is a two-dimensional generalization of the "hat kernel" in (7.14); indeed this function

The parameter z controls the standard distributions in the multivariate Gaussian distribution.

$$h(\mathbf{s}, \mathbf{t}) = \max\left(0, \frac{3}{\pi z^2} \left(1 - \frac{\|\mathbf{s} - \mathbf{t}\|}{z} \right) \right) \quad (8.20)$$

describes a *cone* of radius z. Again, increasing z expands the cone's reach, and more remote values of f contribute to b: thus, more blurring.

Since a right cone of radius z and height h has volume $\pi h z^2/3$, the scaling factor of $3/(\pi z^2) = h$ ensures the cone has volume 1.

8.3.1 *Discretization yields a matrix equation*

As noted earlier, digital images are not functions of continuous variables, but rather are stored as discrete $n \times n$ arrays of pixels, which we regard as a matrix $\mathbf{F} \in \mathbb{R}^{n \times n}$:

$$(\mathbf{F})_{(j,k)} = f_{j,k} = f(t_j, t_k)$$

is the value of the (j,k) pixel, corresponding to the midpoints of the pixel location:

$$t_j = \frac{j - 1/2}{n}, \qquad t_k = \frac{k - 1/2}{n}, \qquad j,k = 1, \ldots, n.$$

Primitive bitmaps use only two colors and hence $f_{j,k}$ only takes two values, 0 or 1. For grayscale images, $f_{j,k}$ is an integer between 0 (black) and 255 (white).

Given that we only know $f(t^{(1)}, t^{(2)})$ at certain discrete values $t^{(1)} = t_j$ and $t^{(2)} = t_k$, we naturally discretize the double integral (8.18) for $b(s^{(1)}, s^{(2)})$ using two applications of the same midpoint rule we used for one-dimensional problem in Section 7.5. Of course, we want to evaluate the blurring function at the same pixel locations use used for the true image, and so we define

$$s_\ell = \frac{\ell - 1/2}{n}, \qquad s_m = \frac{m - 1/2}{n}, \qquad \ell, m = 1, \ldots, n.$$

Thus for $\ell, m = 1, \ldots, n$, we approximate (8.18) via

$$b(s_\ell, s_m) \approx b_{\ell,m} = \frac{1}{n^2} \sum_{j=1}^{n} \sum_{k=1}^{n} h\left(\begin{bmatrix} s_\ell \\ s_m \end{bmatrix}, \begin{bmatrix} t_j \\ t_k \end{bmatrix} \right) f_{j,k}. \tag{8.21}$$

Applying this approximation at each of the n^2 pixels gives n^2 linear equations, which we naturally want to collect into matrix form. To simplify the notation, define

$$h_{j,k,\ell,m} := h\left(\begin{bmatrix} s_\ell \\ s_m \end{bmatrix}, \begin{bmatrix} t_j \\ t_k \end{bmatrix} \right).$$

We have colored the indices here to distinguish those that correspond to the original object pixel value $f_{j,k}$ (in blue) and the blurred pixel value $b_{\ell,m}$ (in red).

With this notation, we can arrange the n^2 equations in (8.21) into matrix form. For a concrete example we illustrate this form for $n = 3$:

$$\begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ \hline b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ \hline b_{1,3} \\ b_{2,3} \\ b_{3,3} \end{bmatrix} = \frac{1}{n^2} \left[\begin{array}{ccc|ccc|ccc} h_{1,1,1,1} & h_{2,1,1,1} & h_{3,1,1,1} & h_{1,2,1,1} & h_{2,2,1,1} & h_{3,2,1,1} & h_{1,3,1,1} & h_{2,3,1,1} & h_{3,3,1,1} \\ h_{1,1,2,1} & h_{2,1,2,1} & h_{3,1,2,1} & h_{1,2,2,1} & h_{2,2,2,1} & h_{3,2,2,1} & h_{1,3,2,1} & h_{2,3,2,1} & h_{3,3,2,1} \\ h_{1,1,3,1} & h_{2,1,3,1} & h_{3,1,3,1} & h_{1,2,3,1} & h_{2,2,3,1} & h_{3,2,3,1} & h_{1,3,3,1} & h_{2,3,3,1} & h_{3,3,3,1} \\ \hline h_{1,1,1,2} & h_{2,1,1,2} & h_{3,1,1,2} & h_{1,2,1,2} & h_{2,2,1,2} & h_{3,2,1,2} & h_{1,3,1,2} & h_{2,3,1,2} & h_{3,3,1,2} \\ h_{1,1,2,2} & h_{2,1,2,2} & h_{3,1,2,2} & h_{1,2,2,2} & h_{2,2,2,2} & h_{3,2,2,2} & h_{1,3,2,2} & h_{2,3,2,2} & h_{3,3,2,2} \\ h_{1,1,3,2} & h_{2,1,3,2} & h_{3,1,3,2} & h_{1,2,3,2} & h_{2,2,3,2} & h_{3,2,3,2} & h_{1,3,3,2} & h_{2,3,3,2} & h_{3,3,3,2} \\ \hline h_{1,1,1,3} & h_{2,1,1,3} & h_{3,1,1,3} & h_{1,2,1,3} & h_{2,2,1,3} & h_{3,2,1,3} & h_{1,3,1,3} & h_{2,3,1,3} & h_{3,3,1,3} \\ h_{1,1,2,3} & h_{2,1,2,3} & h_{3,1,2,3} & h_{1,2,2,3} & h_{2,2,2,3} & h_{3,2,2,3} & h_{1,3,2,3} & h_{2,3,2,3} & h_{3,3,2,3} \\ h_{1,1,3,3} & h_{2,1,3,3} & h_{3,1,3,3} & h_{1,2,3,3} & h_{2,2,3,3} & h_{3,2,3,3} & h_{1,3,3,3} & h_{2,3,3,3} & h_{3,3,3,3} \end{array}\right] \begin{bmatrix} f_{1,1} \\ f_{2,1} \\ f_{3,1} \\ \hline f_{1,2} \\ f_{2,2} \\ f_{3,2} \\ \hline f_{1,3} \\ f_{2,3} \\ f_{3,3} \end{bmatrix}.$$

We write this system as $\mathbf{b} = \mathbf{A}\mathbf{f}$, with $\mathbf{A} \in \mathbb{R}^{n^2 \times n^2}$. Even for a modest number of pixels in each dimension, $\mathbf{A}$ can be quite large: even just *constructing* $\mathbf{A}$ in the obvious way can take an extreme amount of time, calling for slicker algorithms that exploit common properties of blurring kernels. For example, the two kernels in (8.19) and (8.20) only depend on

Make sure you do not forget the $1/n^2$ term in $\mathbf{A}$, an easy mistake to make!

$$\|\mathbf{s} - \mathbf{t}\| = \sqrt{(s_\ell - t_j)^2 + (s_m - t_k)^2} = \frac{1}{n}\sqrt{(\ell - j)^2 + (m - k)^2},$$

so any entries $h_{j,k,\ell,m}$ of $\mathbf{A}$ having the values of $\ell - j$ and $m - k$ will be identical. This fact imparts tremendous structure to the matrix. Indeed, for such kernels the matrix for $n = 3$ will have the structure

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_1^T & \mathbf{A}_0 & \mathbf{A}_1 \\ \mathbf{A}_2^T & \mathbf{A}_1^T & \mathbf{A}_0 \end{bmatrix}, \tag{8.22}$$

where $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2 \in \mathbb{R}^{3\times 3}$ denote the blocks demarcated by the horizontal and vertical lines in the general form of $\mathbf{A}$, above:

- $\mathbf{A}_0$ corresponds to $k - m = 0$;
- $\mathbf{A}_1$ corresponds to $k - m = 1$;
- $\mathbf{A}_2$ corresponds to $k - m = 2$.

The generalization of this structure, called *symmetric block Toeplitz*, to the case of general n should be apparent from this pattern.

To write an efficient code to construct $\mathbf{A} \in \mathbb{R}^{n^2\times n^2}$ for large n, one must exploit this structure.

Figure 8.7 shows the nonzero pattern of the blurring matrix $\mathbf{A}$ using the cone kernel function for several values of n and z. Notice a few properties of these matrices:

- Since the cone blurring function is zero for pixels whose midpoints are further than z apart, many entries in $\mathbf{A}$ are zero. The smaller z, the more zeros. Matrices with many zeros are called *sparse*, and the plots in Figure 8.7 illustrate the *sparsity pattern* of $\mathbf{A}$.

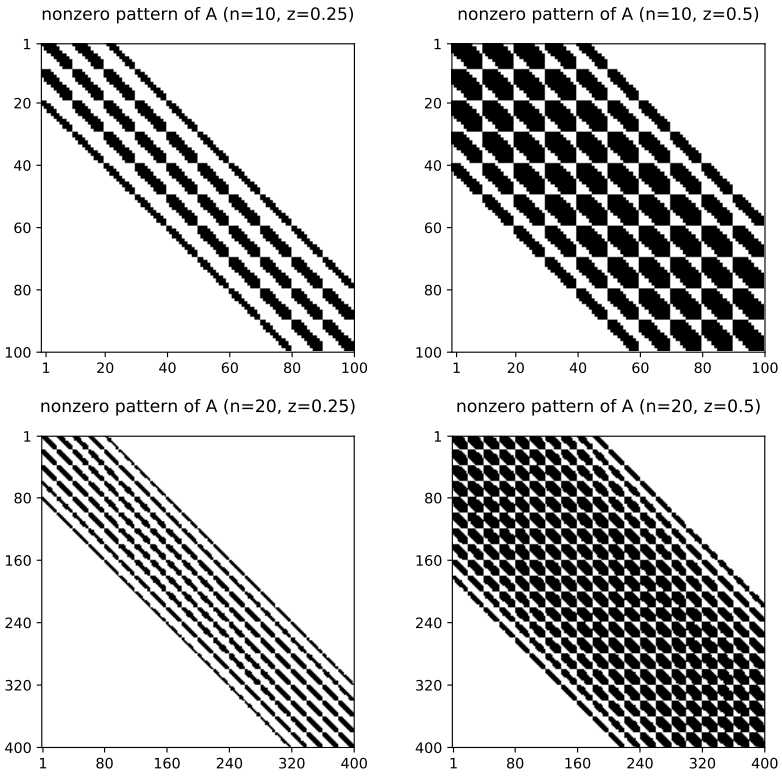


Figure 8.7: Nonzero pattern of the blurring matrix for the cone kernel function, for $n = 10$ and $n = 20$ with $z = 0.25$ and 0.50. As z increases, so does the radius of the cone, and the number of nonzeros in $\mathbf{A}$.

(When building $\mathbf{A}$, one need not waste effort computing these zero entries. For large-scale problems, as we shall discuss in the next chapter, one can exploit this zero structure to get fast algorithms.)

- The nonzero pattern is consistent with the tiling pattern described in (8.22), though these plots do not indicate the values of the entries of $\mathbf{A}$, just the nonzero pattern.
- These matrices are *banded*, meaning they are zero for all diagonals more than the *bandwidth* of $\mathbf{A}$. The bandwidth increases with n and z, although the ratio of the bandwidth to the matrix dimension n^2 stays roughly constant.

8.3.2 *An example of blurring*

Let us see blurring in action, using the simple bitmap shown in Figure 8.8. This image comprises 80×80 pixels, which are assigned values 0 and 1. The blurring matrix $\mathbf{A} \in \mathbb{R}^{n^2 \times n^2}$ thus has $80^2 = 6400$ rows and columns, a large matrix despite the crude pixelation evident in the image.

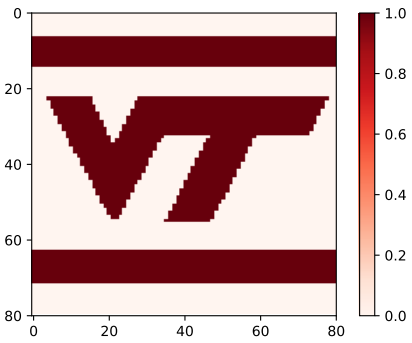


Figure 8.8: A bitmap of 80×80 pixels.

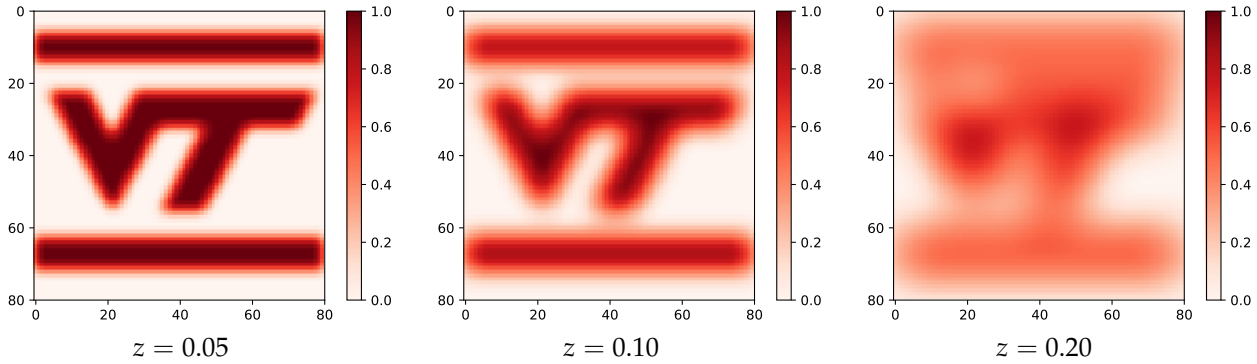


Figure 8.9: Blurred versions of the bitmap in Figure 8.8 using the cone kernel (8.20) with parameters $z = 0.05$, $z = 0.1$, and $z = 0.2$.

Figure 8.9 shows the effect of blurring on this image, using the cone kernel (8.20) with three increasing values of z. Indeed as z increases, the image becomes increasingly difficult to identify. Look at that final image, which used $z = 0.2$. Could you discern that this is a blurry image of Figure 8.8? Do we have any hope of recovering the original image via

$$\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}\,?$$

8.3.3 *A first attempt at deblurring*

Our first attempt to deblur these images will simply apply $\mathbf{A}^{-1}$ to find $\mathbf{f}$ from $\mathbf{b}$. We will start with the "exact" $\mathbf{b}$ (formed via $\mathbf{b}$ =

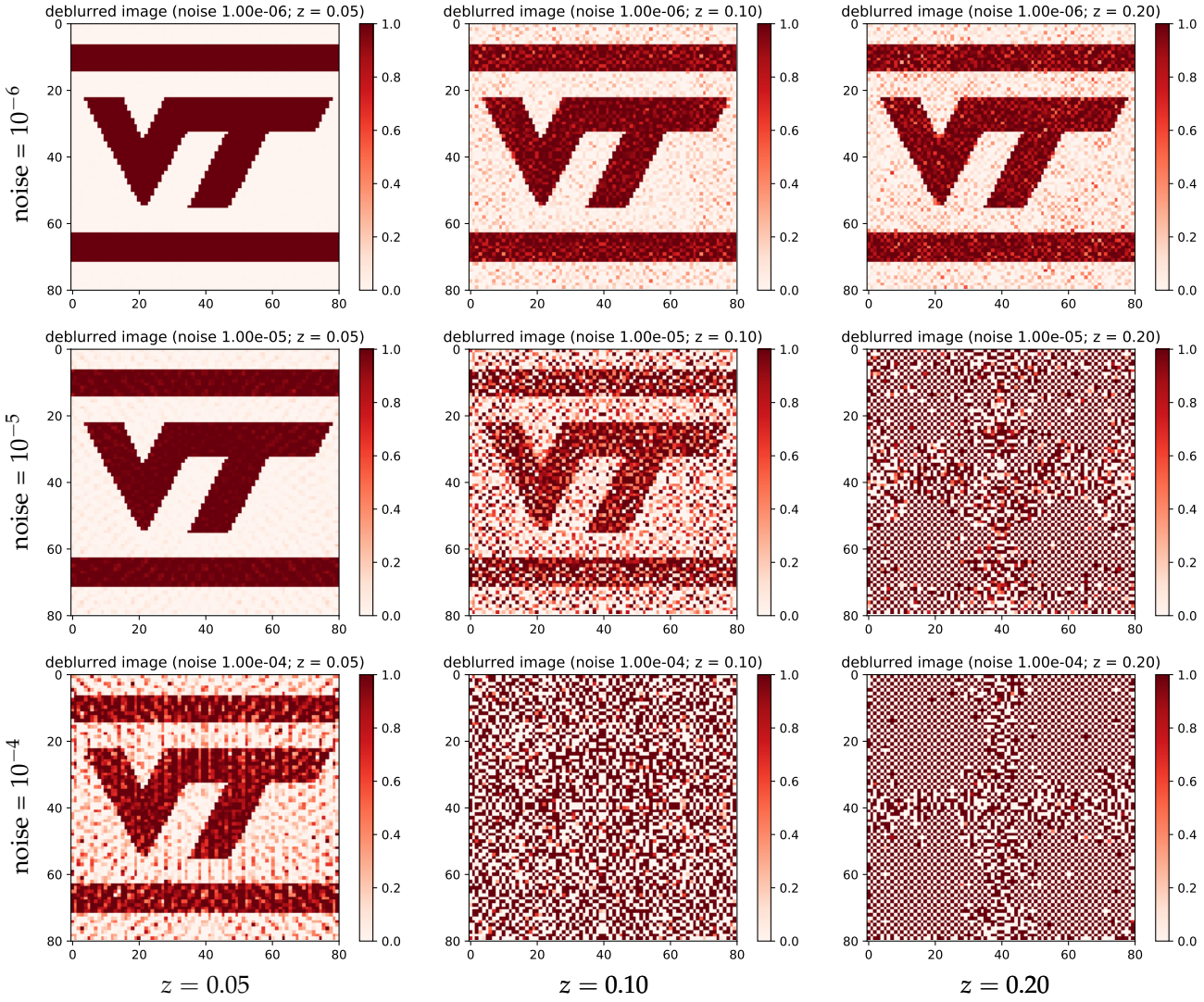


Figure 8.10: Recovered images using $\mathbf{A}^{-1}\mathbf{b}$, where $\mathbf{b}$ has been polluted with Gaussian noise having standard deviation noted in the title of each plot. The blurring gets more severe as z increases, and that has significant implications on the quality with which the images are recovered.

Af, since in this special test case we know the function f we are trying to find) and then add Gaussian random noise with standard deviation 10^{-6}, 10^{-5}, and 10^{-4}. Figure 8.10 shows the results, for increasing amounts of blurring ($z = 0.05$, 0.10, and 0.20. The effect of the noise is obvious in all cases except the top left, which features the least noise (10^{-6}) and the least blurring ($z = 0.05$). As the noise and blurring increases, the recovered **f** no longer resembles the true image in Figure 8.8, but becomes increasingly dominated by *static* or *white noise*.

By now we are not surprised to see such poor "solutions" to an deblurring problem, and you hopefully suspect some culprits:

- The singular values of **A** probably show significant decay, with the latter singular values quite small.
- This singular values probably decay faster as the blurring gets stronger (recall the cartoon in Figure 7.9), consistent with the degradation of the results from left to right in Figure 8.10.
- The right singular vectors $\mathbf{v}_j$ associated with small singular values σ_j probably have some static appearance, as we emerging and then dominating the recovered **f** vectors.

Figure 8.11 shows the singular values of **A** for the three z values we have been studying. Indeed, our instinct is confirmed: the singular values decay seven orders of magnitude, with an initial rapid period of decay, followed by more gradual decay, and then a more rapid tail off at the end. (Note the horizontal axis: we are looking at

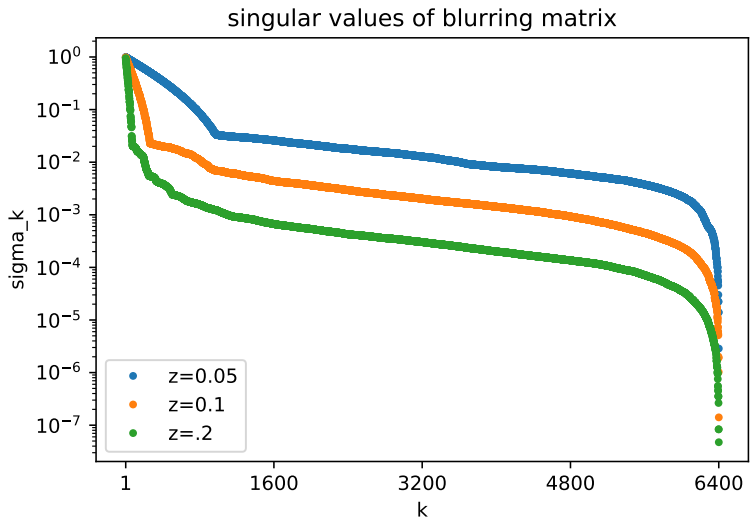


Figure 8.11: Singular values of the blurring matrix for $n = 80$ with the three values of $z = 0.05$, 0.1, and 0.2. As z increases, the matrix blurs more strongly, and the singular values get smaller.

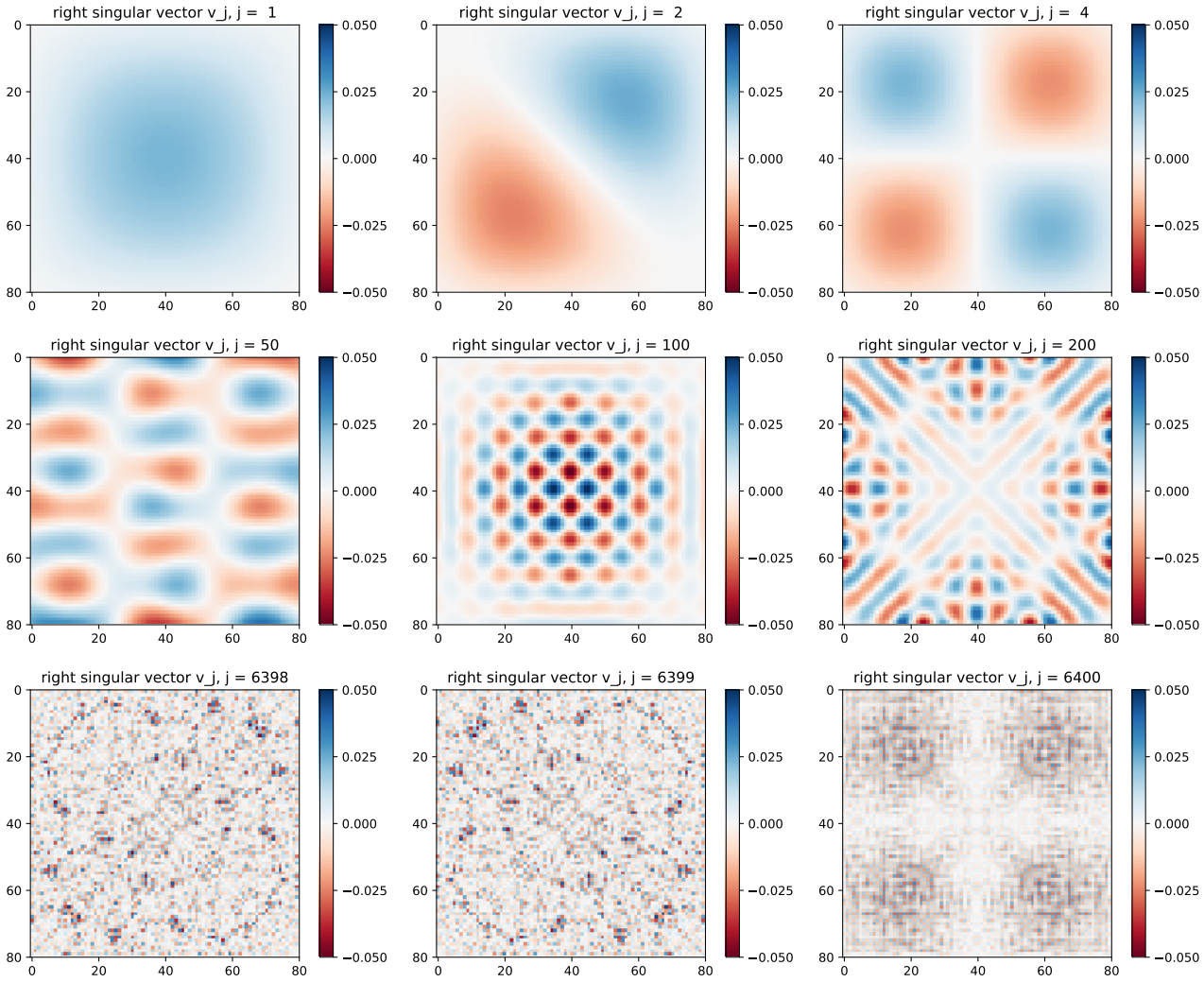


Figure 8.12: Singular vectors of $\mathbf{A}$ for $n = 80$ and $z = 0.2$, $\mathbf{v}_j \in \mathbb{R}^{n^2}$; the vectors have been reshaped into $n \times n$ matrices to facilitate comparison with the images shown in this section. The top rows shows $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_4$ corresponding to large singular values; the middle rows shows $\mathbf{v}_{50}$, $\mathbf{v}_{100}$, and $\mathbf{v}_{200}$ corresponding to middling singular values, and the last row shows $\mathbf{v}_{6398}$, $\mathbf{v}_{6399}$, and $\mathbf{v}_{6400}$ corresponding to the smallest singular values.

$n^2 = 6400$ singular values for each matrix. As z increases and the blur becomes more severe, the singular values get smaller.

Figure 8.12 shows nine of the right singular vectors $\mathbf{v}_j$. These singular vectors are each column vectors of length n^2, but we rearrange them into an $n \times n$ matrix, just as we do for our image vectors, for a deeper understanding of how these singular vectors contribute to the deblurring process. The top row shows early singular vectors ($\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_4$), all of which exhibit little oscillation. The middle row shows intermediate singular vectors ($\mathbf{v}_{50}$, $\mathbf{v}_{100}$, and $\mathbf{v}_{200}$), increasing in frequency of oscillation as the index increases. The final row shows the last three singular vectors ($\mathbf{v}_{6398}$, $\mathbf{v}_{6399}$, and $\mathbf{v}_{6400}$). Indeed, as expected, these singular vectors exhibit the *static* pattern we saw for the worst recovered $\mathbf{f}$ in Figure 8.10.

Figure 8.13 illustrates why our recovery efforts in Figure 8.10 went

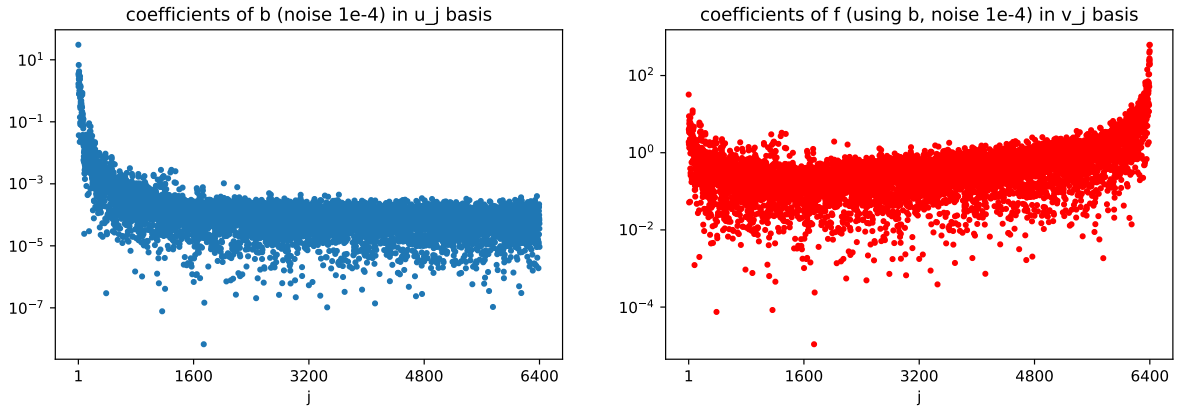


Figure 8.13: Magnitude of coefficients $|\mathbf{u}_j^T\mathbf{b}|$ of $\mathbf{b}$ in the $\{\mathbf{u}_j\}$ basis of left singular vectors, and $|\mathbf{u}_j^T\mathbf{b}/\sigma_j$ of $\mathbf{A}^{-1}\mathbf{b}$, for $\mathbf{b}$ generated with Gaussian noise having standard deviation 10^{-4} and blurring parameter 0.2.

so wrong. We single out the worst of these examples, noise level 10^{-4} with blurring parameter $z = 0.20$. The left plot of Figure 8.13 examines the values $|\mathbf{u}_j^T\mathbf{b}|$ that arise in the expansion of $\mathbf{b}$ in the orthonormal basis of *left* singular vectors:

$$\mathbf{b} = \sum_{j=1}^{n^2} (\mathbf{u}_j^T\mathbf{b})\,\mathbf{u}_j.$$

As we expect, $\mathbf{u}_j^T\mathbf{b}$ is largest for the small j, and eventually falls off roughly to the level of the noise, 10^{-4} that we inflicted upon $\mathbf{b}$.

Contrast that left plot with the one on the right, which shows $|\mathbf{u}_j^T\mathbf{b}|/\sigma_j$, the expansion coefficients of the solution $\mathbf{f}$ in the orthonormal basis of *right* singular vectors

$$\mathbf{f}_+ = \mathbf{A}^+\mathbf{b} = \sum_{j=1}^{n^2} \left(\frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\right)\mathbf{v}_j. \tag{8.23}$$

The left side of that the plot looks fine, as the coefficients for small j dominate. But soon we reach the level where the singular value σ_j decays below the value of $\mathbf{u}_j^T\mathbf{b}$ (which was elevated around 10^{-4} by the noise), and the $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ terms start increasing; eventually they become large enough to overwhelm the good information contained for smaller j values.

8.3.4 *Deblurring with truncated SVD regularization*

Figure 8.13 justifies our first strategy for regularization: from the expansion (8.24) for the recovered solution, omit the terms corresponding to small singular values:

$$\mathbf{f}_k = \sum_{j=1}^{k} \left(\frac{\mathbf{u}_j^T\mathbf{b}}{\sigma_j}\right)\mathbf{v}_j. \tag{8.24}$$

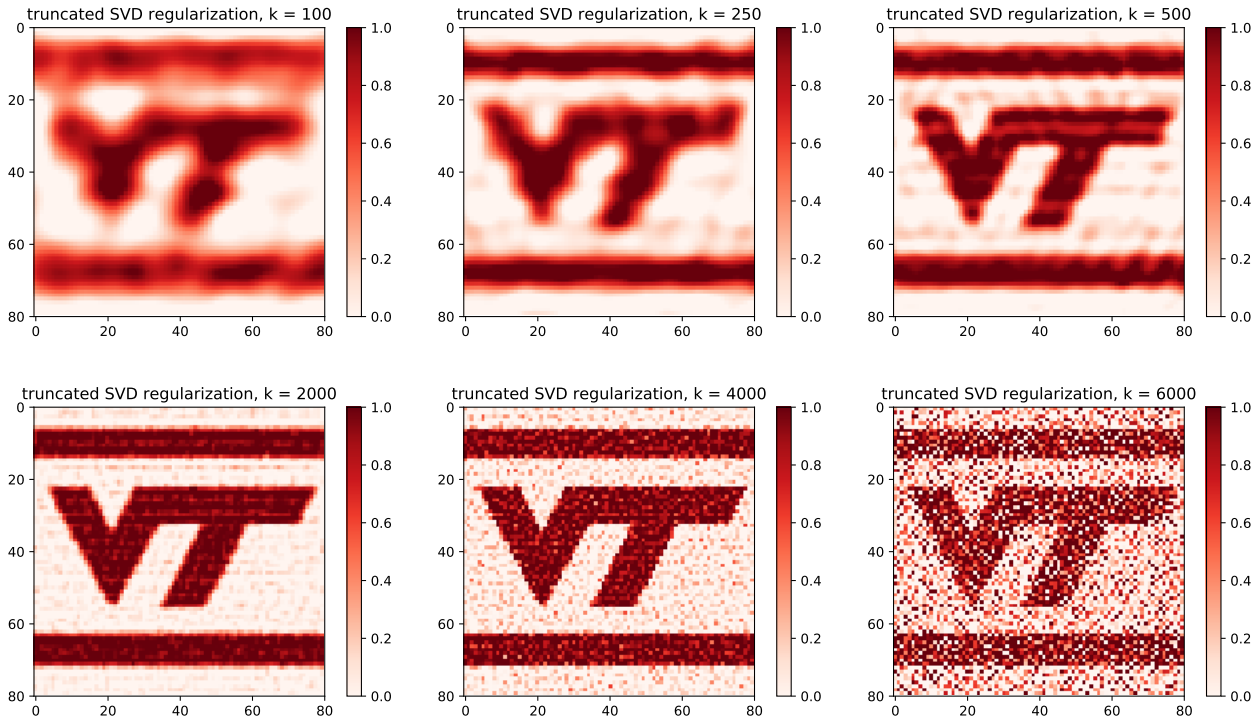


Figure 8.14: Recovery of the case with $z = 0.2$ blurring and noise level 10^{-4} using truncation based on the first k terms of the singular value decomposition.

How does this regularization by truncation to the first k terms of the SVD perform for our example? Figure 8.14 illustrates the results for six values of k.

Calibrate yourself to the dimension of the problem: the matrix $\mathbf{A}$ is $n^2 \times n^2 = 6400 \times 6400$, and the singular values shown in Figure 8.11 do not decay all that quickly. To the eye, the best of these truncated SVD recoveries occurs for $k = 2000$, which might seem excessively large but still includes less than a third of the singular values. If k too small, we do not include enough of the singular vectors with moderate frequency to make out the finer detail in the image; see Figure 8.12. On the other hand, when k gets too large, we include some of the terms for which $\mathbf{u}_j^T\mathbf{b}/\sigma_j$ was problematically large, corresponding to static-looking singular vectors: hence the poor results for $k = 4000$ and $k = 6000$.

Figure 8.15 shows an L-curve for SVD truncation.

8.3.5 Deblurring with Tikhonov regularization

Tikhonov regularization provides similarly strong recoveries,

$$\mathbf{f}_\lambda = \sum_{j=1}^{n^2} \left(\frac{\sigma_j}{\sigma_j^2 + \lambda^2} \mathbf{u}_j^T \mathbf{b} \right) \mathbf{v}_j,$$

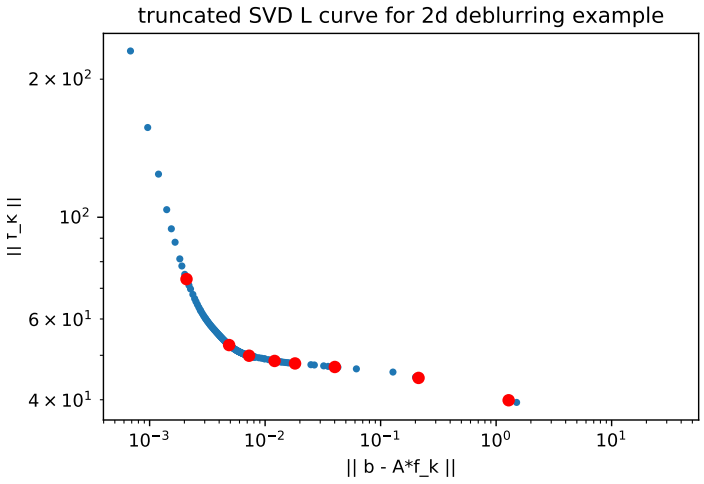


Figure 8.15: L curve for truncated SVD regularization

as shown in Figure 8.16. If λ is too small, the regularization insufficiently damps the problematic singular values, and the static dominates. As λ increases we eventually get excellent recoveries, with $\lambda = 10^{-3}$ giving the best results of those shown here. If λ is too large, then the penalty term $\lambda^2\|\mathbf{f}\|^2$ suppresses the solution and we again get poor recoveries that now look excessively smooth.

Figure 8.17 shows an L-curve for Tikhonov regularization. Our favored solution, with $\lambda = 10^{-3}$, falls just after the bend in the L.

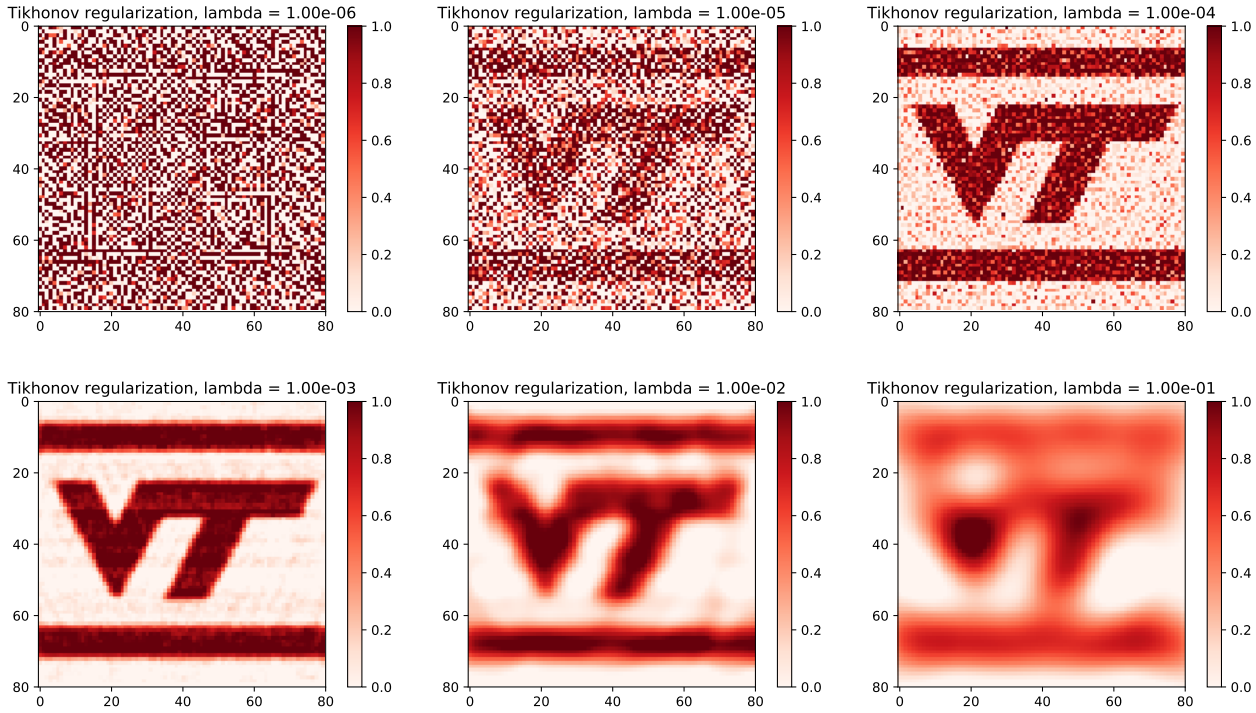


Figure 8.16: Recovery of the case with $z = 0.2$ blurring and noise level 10^{-4} using Tikhonov regularization, for six values of the regularization parameter λ.

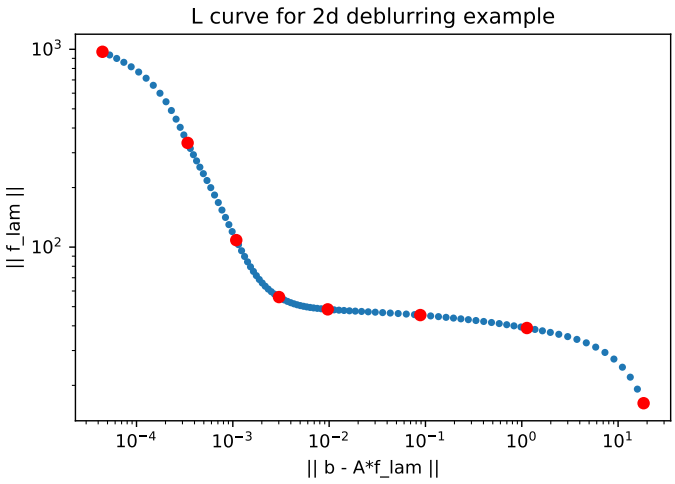
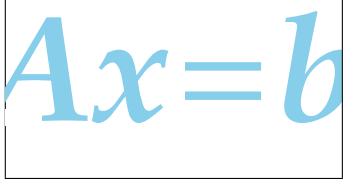


Figure 8.17: L curve for Tikhonov regularization

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 9
Iterative methods for large linear systems

The blurring matrices we constructed in the last chapter for two-dimensional images grew rapidly in dimension as the number of pixels increased. An image comprised of $n \times n$ pixels led to a blurring matrix $\mathbf{A}$ of dimension $n^2 \times n^2$, so that our example with $n = 80$ gave 6400×6400 matrices. As matrices grow to such dimensions, computations no longer appear to happen instantly; distinctions between the performance of algorithms, which might seem of academic interest when $\mathbf{A}$ is of small dimension, suddenly become much more acute: the difference between methods that take 1 minute (or 1 day) versus 2 minutes (or 2 days) is rather stark.

Matrices of dimension 6400×6400 start to approach the limit at which we can comfortably compute with generic algorithms on modern laptops. By *exploiting structure* in $\mathbf{A}$, one can often obtain results much more quickly, though the algorithms grow in sophistication and, potentially, numerical robustness.

9.1 Sparse matrices

Various structures are worth exploiting. Symmetry is the most common and simple property worth addressing, leading to significant speedups for solving $\mathbf{Ax} = \mathbf{b}$ and computing eigenvalues. Beyond symmetry, the most important structure is *sparsity*. The definition of a *sparse matrix* will be a bit fuzzier than our other definitions in these notes. This famous numerical analyst James Hardy Wilkinson informally defined a *sparse matrix* as "any matrix with enough zeros that it pays to take advantage of them." Implicit in this definition is that some algorithms are quite sensitive to the *location* of these nonzero entries, while others are relatively immune to that consideration.

Quoted on page 334 of J. R. Gilbert, C. Moler, R. Schreiber, "Sparse Matrices in MATLAB: Design and Implementation," *SIAM J. Matrix Anal. Appl.* 13 (1992) 333–356.

We give a less punchy version of Wilkinson's definition.

Definition 9.1. *A matrix is* sparse *provided it has sufficiently many zero entries to be worth exploiting by an algorithm. This assessment depends not only on the number of nonzero entries, but also on* their location *in* **A** *and how the algorithm operates upon* **A**.

A matrix that is not sparse is called dense.

Figure 9.1 illustrates two 50×50 matrices that have the same number of nonzeros, but distributed differently throughout the matrix. The top pattern is ideal: we can use compact storage and every algorithm can take advantage of this structure. The second matrix, with its nonzeros scattered throughout the matrix, requires a more general sparse matrix data structure, and its structure is not favorable to all algorithms. Despite that fact that it has so many zero entries, some algorithms could be better off treating this as a dense matrix.

We have already encountered sparse matrices in these notes: Figure 8.7 showed the nonzero pattern for four blurring matrices. In those cases, the nonzero entries correspond to the pixels that influence the blurring of a given pixel: the larger the blurring parameter z, the more nonzero entries.

To further appreciate the origins of sparse matrices, we introduce an antique example. In the late 16th century, the Oxford mathematician John Wallis (1616–1703) showed how one could build a flat roof made up of interlocking short timbers. Figure 9.2 shows his design. To compute his design, Wallis developed a system of 25 linear equations in 25 unknowns (with one free variable, T, a scaling factor; set $T = 1$ to obtain the right-hand side).

Figure 9.3 shows these 25 equations. Notice that each of the equations only involves a couple of variables, *reflecting the connections of the beams* in Figure 9.2. Setting the free variable $T = 1$, the first equa-

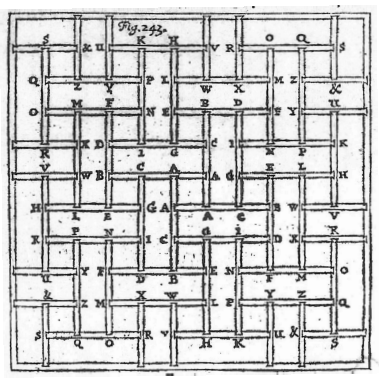


Figure 9.2: Wallis's design for a flat roof made of short interlocking timbers, from page 953 of the first volume of his *Opera Mathematica*, 1695.

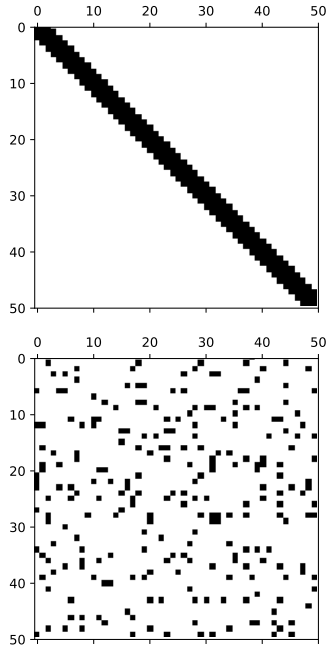


Figure 9.1: Location of nonzero entries (the *sparsity pattern*) of two 50×50 sparse matrices, both having 244 nonzero entries. The top example, a *banded matrix*, has structure that is easy to store and exploit in an algorithm; the bottom example, with nonzeros scattered arbitrarily throughout the matrix, takes more overhead to store and is hard for some algorithms to exploit.

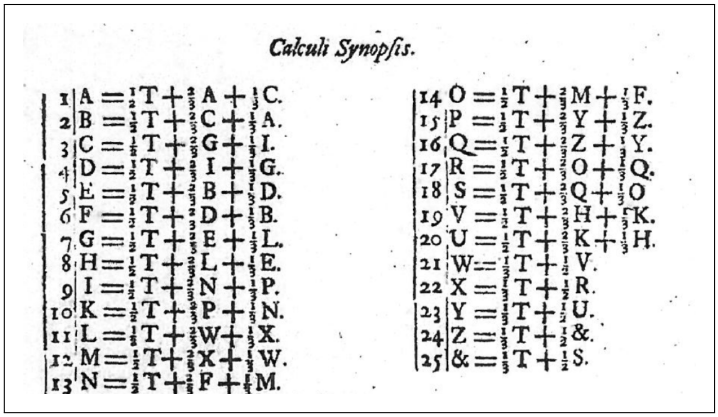


Figure 9.3: Wallis's system of 25 linear equations for the design of his flat roof, in terms of the common scaling factor T.

tion $A = \frac{1}{2}T + \frac{2}{3}A + \frac{1}{3}C$ can be rearranged as

$$\tfrac{1}{3}A - \tfrac{1}{3}C = \tfrac{1}{2}. \tag{9.1}$$

Wallis was working more than 150 years before the development of matrix technology; remarkably, he tackled his system by simplifying individual equations and substituting. (Yes, he got the answer perfectly correct – an amazing feat!) Had he written his system in a matrix form, Wallis would have obtained the sparse matrix shown in Figure 9.4. Effectively he solved his system using a kind of sparse elimination.

We include this antique example not merely for historical interest, but because it illustrates a more general idea:

sparsity reflects locality.

Whether the matrix describes the blurring of an image, the diffusion of heat, or communication across a social network, when the primary effects are localized, many entries of the matrix (especially those describing the interaction of distant elements of the system) are typically zero.

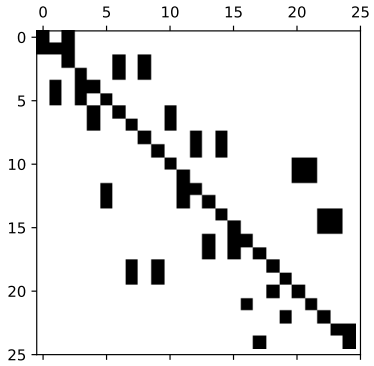


Figure 9.4: Location of nonzero entries (the *sparsity pattern*) of $\mathbf{A}$ for Wallis's model of a flat roof of short interlocking timbers. Only 69 of the $25^2 = 625$ entries in $\mathbf{A}$ are nonzero, an average of less than 3 nonzeros per row.

9.1.1 *Dense direct solvers*

Thus far we have implicitly been solving $\mathbf{Ax} = \mathbf{b}$ using the algorithms of *dense linear algebra*. All n^2 entries of the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ are stored (including zeros), and the algorithms applied to $\mathbf{A}$ do not check whether the entries they are manipulating are zero or not. The most broadly applied of these algorithms require $\mathcal{O}(n^3)$ operations to solve $\mathbf{Ax} = \mathbf{b}$. The $\mathcal{O}(n^2)$ storage and $\mathcal{O}(n^3)$ operations severely limit the size of n for which such methods are viable; typical modern computers can handle $n \leq 5000$ without much trouble, and $n \leq 10000$ in a stretch. Even for such n, and certainly for larger n, algorithms that

exploit matrix structure (such as sparsity) typically perform much better.

9.1.2 *Sparse direct solvers*

Traditional dense matrix algorithms can be adapted to exploit sparsity in $\mathbf{A}$. Sometimes such modifications are easy to implement: for the banded structure seen in the top of Figure 9.1, one can solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ using $\mathcal{O}(n)$ storage $\mathcal{O}(n)$ operations (provided the bandwidth is fixed as n grows), a remarkable speedup over the dense approach. Other modifications are rather more intricate, and are the subject of active research: arbitrary sparsity patterns like the one in the bottom of Figure 9.1 pose considerable challenge. These methods are somewhat invasive, in that they closely manipulate the entries in $\mathbf{A}$; as the algorithm proceeds, some entries of $\mathbf{A}$ that are zero (and hence not stored) need to be *filled in* due to the arithmetic of Gaussian elimination. Such fill-in requires additional storage; how much depends on the structure of $\mathbf{A}$. In some cases the approaches numerical analysts use to preserve *numerical stability* (such as pivoting in Gaussian elimination) work against the goal of *preserving sparsity*, and one must strike a reasonable balance.

For many moderate size problems (n in the tens or hundreds of thousands), *sparse direct methods* can solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ very efficiently. Software implementations of these sophisticated variants of Gaussian elimination are widely available and are the method of choice for solving many $\mathbf{A}\mathbf{x} = \mathbf{b}$. Details are beyond the scope of these notes, but we point interested students toward UMFPACK (Unsymmetric Multi-Frontal PACKage), a leading implementation developed by Tim Davis. One can access a sparse direct solver in Python via

```
scipy.sparse.linalg.spsolve
```

which can also serve as an interface to UMFPACK (provided you have installed that solver).

9.1.3 *Iterative methods*

Iterative methods attempt to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ in a rather less invasive manner than sparse direct methods. In many cases $\mathbf{A}$ is only accessed through the matrix-vector product operation (often called a *matvec*). The kth iteration of the method generates an approximation $\mathbf{x}_k$ to the exact solution, with (hopefully) $\mathbf{x}_k \to \mathbf{x}$ as $k \to \infty$.

We highlight some key advantages of iterative methods.

- A user need only simply a subroutine to compute the matvec $\mathbf{A}\mathbf{v}$ for a given $\mathbf{v}$. *The matrix* $\mathbf{A}$ *does not even need to be created or stored*, provided you have some other way to compute $\mathbf{A}\mathbf{v}$. This feature

makes iterative methods particularly compelling for large-scale problems, where storing $\mathbf{A}$ can be very expensive.

- Iterative methods are generally *easy to implement on parallel computers*. Typically one distributes blocks of n/p rows to each of p different processors.
- After a modest number of iterations k, the iterate $\mathbf{x}_k$ often approximates the true solution $\mathbf{x}$ to a few digits of accuracy. In many applications this limited accuracy is all that is needed. Direct methods (be they sparse or dense) provide *no digits of accuracy* until the very last stage of the computation, where they deliver $\mathbf{x}$ (up to numerical errors).

Iterative methods are not a panacea: they often require some choice of parameters to ensure convergence, and the rate of convergence can be agonizingly slow in some cases; indeed some methods even diverge.

We consider two main classes of iterative methods for $\mathbf{A}\mathbf{x} = \mathbf{b}$.

- *Splitting methods* (also called *stationary iterative methods* express $\mathbf{A}$ as the sum of two pieces,

  $$\mathbf{A} = \mathbf{M} - \mathbf{N},$$

  where $\mathbf{M}$ is much easier to invert than $\mathbf{A}$. The iteration then proceeds as

  $$\mathbf{x}_{k+1} = \mathbf{M}^{-1}\big(\mathbf{N}\mathbf{x}_k + \mathbf{b}\big).$$

  Convergence is measured through the residual

  $$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k,$$

  which one can show to obey the formula

  $$\mathbf{r}_k = (\mathbf{N}\mathbf{M}^{-1})^k\, \mathbf{r}_0.$$

  The method will converge provided all eigenvalues of the iteration matrix $\mathbf{N}\mathbf{M}^{-1}$ are smaller than one in magnitude. Unfortunately, knowing whether this condition will hold for a given matrix $\mathbf{A}$ and a given splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is often unclear. This uncertainty restricts the utility of splitting methods, and has caused them to fall from favor as general-purpose solvers in recent years, though they remain important tools in certain domains (e.g., *multigrid* algorithms).

  For example, choosing $\mathbf{M}$ to be the main diagonal of $\mathbf{A}$ yields the Jacobi method; other popular iterations, the Gauss–Seidel and SOR methods, take $\mathbf{M}$ to be upper triangular.

- *Polynomial iterative methods* express $\mathbf{x}_k$ in the form

  $$\mathbf{x}_k = q_k(\mathbf{A})\mathbf{b},$$

where q_k is a polynomial of degree less than k. Such methods are the focus of the next section of these notes. As we shall see, there exists a good approach for selecting an optimal q_k that ensures convergence, though construction of this q_k can be expensive. Convergence can often be accelerated by choosing q_k in a faster, sub-optimal way, or modifying the problem (using a technique called *preconditioning*; see Section 9.8) to make it better suited for fast convergence.

9.2 *Polynomial iterative methods: abstract framework*

At their kth step, polynomial iterative methods produce an approximation $\mathbf{x}_k \in \mathbb{R}^n$ to the true solution $\mathbf{x} \in \mathbb{R}^n$ of the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ having the form

$$\mathbf{x}_k = q_k(\mathbf{A})\mathbf{b}, \tag{9.2}$$

Important note: these methods do not actually *construct* $\mathbf{x}_k$ by evaluating $q_k(\mathbf{A})\mathbf{b}$ at every k. They build up $\mathbf{x}_k$ in a much more efficient manner, but this formula for $\mathbf{x}_k$ provides a framework for organizing, comparing and analyzing these methods.

where q_k is a polynomial of degree less than k.

To assess convergence, one might naturally want to monitor the *error vector*

$$\mathbf{e}_k := \mathbf{x} - \mathbf{x}_k.$$

However, $\mathbf{e}_k$ is not available: if we knew $\mathbf{e}_k$ and $\mathbf{x}_k$, we could simply create the true solution, $\mathbf{x} = \mathbf{e}_k + \mathbf{x}_k$. Instead, we gauge convergence through the *residual vector*

$$\|Br_k := \mathbf{b} - \mathbf{A}\mathbf{x}_k.$$

Notice that $\mathbf{A}\mathbf{e}_k = \mathbf{r}_k$, so if $\|\mathbf{r}_k\| \to 0$, then $\|\mathbf{e}_k\| \to 0$.

The residual can also be characterized using polynomials. Using the polynomial expression (9.2) for $\mathbf{x}_k$,

$$\begin{aligned}
\mathbf{r}_k &:= \mathbf{b} - \mathbf{A}\mathbf{x}_k \\
&= \mathbf{b} - \mathbf{A}q_k(\mathbf{A})\mathbf{b} \\
&= \big(\mathbf{I} - \mathbf{A}\mathbf{q}_k(\mathbf{A})\big)\mathbf{b} \\
&= p_k(\mathbf{A})\mathbf{b},
\end{aligned}$$

where we have introduced the *residual polynomial*

$$p_k(z) := 1 - zq_k(z).$$

Here z denotes a generic scalar variable, just used to express the polynomial. Notice that the form of p_k ensure that $p_k(0) = 1 - 0 \cdot q_k(0) = 1$.

The kth step of a polynomial iterative method constructs an approximation $\mathbf{x}_k$ to the solution $\mathbf{x}$ of the form

$$\mathbf{x}_k = q_k(\mathbf{A})\mathbf{b},$$

where the *iteration polynomial* q_k has degree less than k.

The residual $\mathbf{r}_k := \mathbf{b} - \mathbf{A}\mathbf{x}_k$ can be written as

$$\mathbf{r}_k = p_k(\mathbf{A})\mathbf{b},$$

where the *residual polynomial*

$$p_k(z) = 1 - zq_k(z)$$

has degree k or less and satisfies

$$P_k(0) = 1.$$

While still remaining in this highly abstract framework, we can readily develop a general-purpose convergence bound for polynomial iterative methods that will serve us well in the rest of the chapter. For simplicity, we shall assume that $\mathbf{A}$ is a *symmetric* matrix. Polynomial iterative methods certainly also apply to nonsymmetric matrices, but the convergence theory is quite a bit more subtle and remains an area of active research.

Suppose $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a nonsingular *symmetric* matrix. Then, as detailed in Chapter 4, the matrix $\mathbf{A}$ can be written in the diagonalized form

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \mathbf{v}_j^T,$$

where the columns of $\mathbf{V} \in \mathbb{R}^{n\times n}$ are the orthonormal eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ associated with the eigenvalues $\lambda_1, \ldots, \lambda_n$; the matrix $\mathbf{\Lambda} \in \mathbb{R}^{n\times n}$ has the eigenvalues on the main diagonal, and zero in all other entries.

Since $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ for the square matrix $\mathbf{V} \in \mathbb{R}^{n\times n}$, we must have that $\mathbf{V}^{-1} = \mathbf{V}^T$, and hence $\mathbf{V}\mathbf{V}^T = \mathbf{I}$. Thus we can write

$$\mathbf{b} = \mathbf{V}\mathbf{V}^T\mathbf{b} = \mathbf{V}(\mathbf{V}^T\mathbf{b}) = \sum_{i=1}^{n} (\mathbf{v}_j^T\mathbf{b})\mathbf{v}_j,$$

The numbers $\mathbf{v}_j^T\mathbf{b}$ are the coefficients of $\mathbf{b}$ expanded in the basis $\{\mathbf{v}_j\}$ for $\mathbb{R}^n$ formed by the eigenvectors of $\mathbf{A}$.

from which it follows (by the Pythagorean Theorem) that

$$\|\mathbf{b}\|^2 = \sum_{j=1}^{n} (\mathbf{v}_j^T\mathbf{b})^2. \tag{9.3}$$

The iteration polynomial evaluated at $\mathbf{A}$ has a nice form in terms

of the eigenvalue decomposition of $\mathbf{A}$:

$$q_k(\mathbf{A}) = \sum_{j=1}^{n} q_k(\lambda_j)\mathbf{v}_j\mathbf{v}_j^T.$$

This form also leads to a nice expression for the residual polynomial evaluated at $\mathbf{A}$:

$$\mathbf{I} - \mathbf{A}q_k(\mathbf{A}) = p_k(\mathbf{A}) = \sum_{j=1}^{n} p_k(\lambda_j)\mathbf{v}_j\mathbf{v}_j^T$$

With this last expressions in mind, we can develop a bound on the norm of the residual $r_k := \mathbf{b} - \mathbf{A}\mathbf{x}_k$:

$$\begin{aligned}
\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|^2 = \|\mathbf{r}_k\|^2 &= \|p_k(\mathbf{A})\mathbf{b}\|^2 \\
&= \left\| \sum_{j=1}^{n} p_k(\lambda_j)(\mathbf{v}_j^T\mathbf{b})\mathbf{v}_j \right\|^2 \\
&= \sum_{j=1}^{n} \big(p_k(\lambda_j)\big)^2(\mathbf{v}_j^T\mathbf{b})^2 \\
&\leq \Big(\max_{1\leq j\leq n} \big(p_k(\lambda_j)\big)^2 \Big) \sum_{j=1}^{n} (\mathbf{v}_j^T\mathbf{b})^2 \\
&= \Big(\max_{1\leq j\leq n} \big(p_k(\lambda_j)\big)^2 \Big) \|\mathbf{b}\|^2,
\end{aligned}$$

where this last step used (9.3). Dividing by $\|\mathbf{b}\|^2$ and taking square roots, we obtain the error bound

$$\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}\|} \leq \max_{1\leq j\leq n} |p_k(\lambda_j)|.$$

Theorem 9.1. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a nonsingular symmetric matrix having (real) eigenvalues $\lambda_1, \ldots, \lambda_n$. Then the residual $\mathbf{r}_k$ induced by the kth iterate $\mathbf{x}_k = q_k(\mathbf{A})\mathbf{b}$ of the polynomial iterative method satisfies the bound*

$$\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}\|} \leq \max_{1\leq j\leq n} |p_k(\lambda_j)|,$$

for the residual polynomial $p_k(z) = 1 - zq_k(z)$.

9.3 *Richardson's method*

Having set the abstract framework for polynomial iterative methods, it is time to get down to precise methods. The simplest such

algorithm, attributed to Lewis Fry Richardson (1881–1953), just updates the iterate $\mathbf{x}_k$ with a fixed multiple $c \in \mathbb{R}$ of the residual:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + c\mathbf{r}_k. \tag{9.4}$$

This expression does not immediately suggest that Richardson's method fits the template for a polynomial iterative method detailed in the last section. A few steps of the method will make that polynomial nature apparent. Start from $\mathbf{x}_0 = \mathbf{0}$, so that $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b}$, and compute

$$\mathbf{x}_1 = \mathbf{x}_0 + c\mathbf{r}_0 = c\mathbf{b}, \qquad \mathbf{r}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_1 = (\mathbf{I} - c\mathbf{A})\mathbf{b};$$

$$\begin{aligned}\mathbf{x}_2 &= \mathbf{x}_1 + c\mathbf{r}_1 = c\mathbf{b} + c(\mathbf{I} - c\mathbf{A})\mathbf{b} \\ &= (2c\mathbf{I} - c^2\mathbf{A})\mathbf{b},\end{aligned} \qquad \mathbf{r}_2 = \mathbf{b} - \mathbf{A}\mathbf{x}_2 = (\mathbf{I} - c\mathbf{A})^2\mathbf{b};$$

$$\begin{aligned}\mathbf{x}_3 &= \mathbf{x}_2 + c\mathbf{r}_2 \\ &= c\mathbf{b} + c(\mathbf{I} - c\mathbf{A})\mathbf{b} + c(\mathbf{I} - c\mathbf{A})^2\mathbf{b} \\ &= (3c\mathbf{I} - 3c^2\mathbf{A} + c^3\mathbf{A}^2)\mathbf{b},\end{aligned} \qquad \mathbf{r}_3 = \mathbf{b} - \mathbf{A}\mathbf{x}_3 = (\mathbf{I} - c\mathbf{A})^3\mathbf{b}.$$

These few iterations reveal the structure of the iteration and residual polynomials. For example, for $k = 3$ the iteration polynomial is

$$q_k(z) = 3c - 3c^2z + c^3z^2,$$

while the residual polynomial is

$$p_k(z) = 1 - zq(z) = (1 - cz)^3.$$

Indeed, for Richardson's method the residual polynomial always follows this simple form.

For Richardson's method with fixed parameter $c \in \mathbb{R}$, the residual polynomial p_k at step k takes the form

$$p_k(z) = (1 - cz)^k. \tag{9.5}$$

Example 9.1. Iterative methods are meant for large $\mathbf{A}$, but we can often get good insight about their behavior from small examples. In that spirit, we will apply Richardson's method to

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \tag{9.6}$$

corresponding to the exact solution

$$\mathbf{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Start with the initial guess $\mathbf{x}_0 = \mathbf{0}$, and set the iteration parameter to $c = 1/2$. Apply the iteration (9.4) to compute a few iterates and residuals.

$$\mathbf{x}_1 = \begin{bmatrix} 1/2 \\ 3/2 \end{bmatrix} \qquad \mathbf{r}_1 = \begin{bmatrix} 1/2 \\ -3/2 \end{bmatrix}$$

$$\mathbf{x}_2 = \begin{bmatrix} 3/4 \\ 3/4 \end{bmatrix} \qquad \mathbf{r}_2 = \begin{bmatrix} 1/4 \\ 3/4 \end{bmatrix}$$

$$\mathbf{x}_3 = \begin{bmatrix} 7/8 \\ 9/8 \end{bmatrix} \qquad \mathbf{r}_3 = \begin{bmatrix} 1/8 \\ -3/8 \end{bmatrix}$$

$$\mathbf{x}_4 = \begin{bmatrix} 15/16 \\ 15/16 \end{bmatrix} \qquad \mathbf{r}_4 = \begin{bmatrix} 1/16 \\ 3/16 \end{bmatrix}$$

The pattern is evident: $\mathbf{x}_k \to \mathbf{x}$ while $\mathbf{r}_k \to \mathbf{0}$ as $k \to \infty$. (Indeed, notice that the residual is perfectly *cut in half* at each step: $\|\mathbf{r}_{k+1}\| = \frac{1}{2}\|\mathbf{r}_k\|$ for this tidy example.)

The parameter $c = 1/2$ played a crucial role in this convergence. This parameter dictates the influence of the residual $\mathbf{r}_k$ on the new iterate $\mathbf{x}_{k+1}$ according to the update formula $\mathbf{x}_{k+1} = \mathbf{x}_k + c\mathbf{r}_k$. Small values of c restrict the influence of $\mathbf{r}_k$, leading to slow convergence. However, if c is too large, $\mathbf{r}_k$ dominates $\mathbf{x}_{k+1}$, and the iteration can even *diverge*. Figure 9.5 illustrates how the convergence behavior depends on c.

We seek to understand how c controls convergence, to give some guidance on how to select this parameter to optimize the convergence rate.

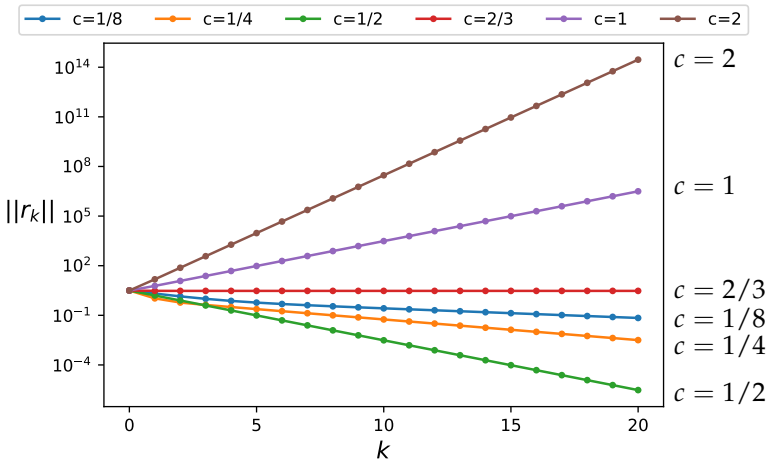


Figure 9.5: Evolution of $\|\mathbf{r}_k\|$ as k increases for six choices of the iteration parameter c in RICHARDSON's method. The choice $c = 1/2$ delivers the best convergence; convergence slows as c is decreased from this optimal value; increasing c also leads to slower convergence, or even *divergence* when $c > 2/3$.

9.3.1 *Convergence of Richardson's method*

We can understand the convergence of RICHARDSON's method (for symmetric $\mathbf{A}$) by simply combining the general convergence result for polynomial iterative methods in Theorem 9.1 with the formula (9.5) for the residual polynomial for RICHARDSON's method, yielding

$$\frac{\|\mathbf{r}_k\|}{\|B\mathbf{b}\|} \leq \max_{1\leq j\leq n} |1 - c\lambda_j|^k. \tag{9.7}$$

The *rate of convergence* for this method,

$$\rho(c) := \max_{1\leq j\leq n} |1 - c\lambda_j|, \tag{9.8}$$

describes the fraction be which we expect the residual norm to decrease at every step:

$$\|\mathbf{r}_{k+1}\| \approx \rho(c)\,\|\mathbf{r}_k\|.$$

To get the fastest overall convergence, we seek the value of c that makes $\rho(c)$ as small as possible.

The plots in Figure 9.6 illustrate the residual polynomials $p_k(z)$ for the first four iterations $k = 1, 2, 3, 4$, given four choices of c, for the matrix (9.6) in Example 9.1. We can label the eigenvalues of $\mathbf{A}$ as

$$\lambda_1 = 3, \qquad \lambda_2 = 1.$$

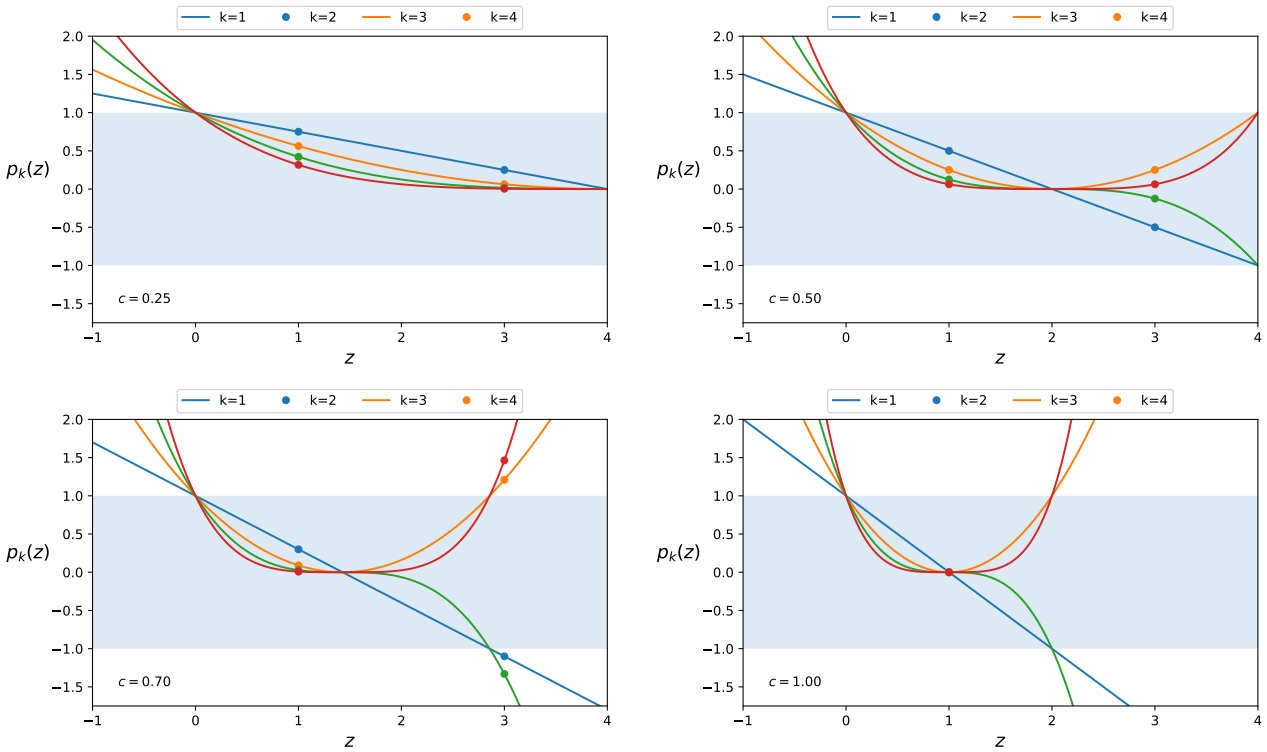


Figure 9.6: Residual polynomial p_k for $k = 1, \ldots, 4$ for Richardson's method applied to the problem in Example 9.1 with four values of the parameter c.

Each illustration includes a shaded band highlighting when $|p(z)| < 1$, and shows dots at $p_k(\lambda_j)$. For convergence, we seek to drive $p_k(\lambda_j) \to 0$ as rapidly as possible as k increases. Consider these four cases.

- When $c = 0.25$, the $p_k(1)$ is slow to converge, but $p_k(3)$ is quite small.
- When $c = 0.50$, $p_k(1) = -p_k(3)$: the magnitude of the polynomial is perfectly balanced between the two eigenvalues, and convergence is rapid and even.
- When $c = 0.70$, $p_k(1)$ gets small quicker than it did for $c = 0.50$, but now $p_k(3)$ actually *increases*: the iteration *diverges* as $k \to \infty$. Note that convergence rate $\rho(0.70) > 1$.
- When $c = 1.00$, $p_k(1) = 0$, but $p_k(3)$ is so large that these points do not even show up on the plot. The method *diverges quickly*.

These illustrations signal a general principle for selecting the parameter c for a matrix $\mathbf{A}$ with positive real eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0.$$

- The optimal value of $c_\star$ will balance $1 - c\lambda_j$ at the extreme eigenvalues of $\mathbf{A}$:
$$1 - c_\star \lambda_n = -(1 - c_\star \lambda_1).$$
- If $0 < c < c_\star$, the method converges, but slower than with $c_\star$.
- If $c_\star < c < 2/\lambda_1$, the method converges, but slower than with $c_\star$.
- If $c_\star > 2/\lambda_1$, the method will diverge for most $\mathbf{b}$.

We conclude this section by describing some advantages and disadvantages of Richardson's method.

+ The algorithm is easy to implement.
+ It only accesses $\mathbf{A}$ through the matrix-vector product $\mathbf{A}\mathbf{x}_k$.
− One needs to know the eigenvalues of $\mathbf{A}$ to pick c optimally.
− Convergence will be very slow when the eigenvalues of $\mathbf{A}$ are spread over a large range.
− The iteration and residual polynomials have a rigid form, e.g.,
$$p_k(z) = (1 - cz)^k.$$

A more flexible method would allow the parameter c to change at each step, giving

$$p_k(z) = (1 - c_1 z)(1 - c_2 z) \cdots (1 - c_k z).$$

With such flexibility we could spread the c_j values so that $1/c_j$ cover the eigenvalues of $\mathbf{A}$ in some way. Indeed, the GMRES algorithm will give this ability, and implicitly select these c_j parameters without requiring knowledge of the eigenvalues of $\mathbf{A}$.

While RICHARDSON's method gives a helpful introduction to polynomial iterative methods, it is not generally regarded as a competitive algorithm, given the alternatives described below.

9.4 *Kaczmarz's Method*

See the *SIAM Review* article by Yair Censor (1981).

9.5 *GMRES*

RICHARDSON's method give a very simple iteration with a rigid residual polynomial form, $p_k(z) = (1 - cz)^k$. We now explore a method that permits a more general form for p_k. The GMRES – Generalized Minimum RESidual – algorithm, swings for the fences: why not use the *best* residual polynomial p_z? GMRES seeks the polynomial of degree k (or less) satisfying the normalization condition $p_k(0) = 1$ that minimizes

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\| = \|\mathbf{r}_k\| = \|p_k(\mathbf{A})\mathbf{b}\|;$$

that is,

$$\|\mathbf{r}_k\| = \min_{\substack{\deg(p) \le k \\ p(0)=1}} \|p(\mathbf{A})\mathbf{b}\| = \|p_k(\mathbf{A})\mathbf{b}\|.$$

Remarkably, we can find this optimal p_k by solving a least squares problem. Let us derive this idea in an idealistic setting, before discussing a practical implementation.

9.5.1 *A theoretical derivation*

At step k, every polynomial iterative method builds an iterate Bx_k having the form

$$\mathbf{x}_k = q_k(\mathbf{A})\mathbf{b},$$

where q_k is a polynomial of degree less than k. Let us write this polynomial in the form

$$q_k(z) = s_0 + s_1 z + s_2 z^2 + \cdots + s_{k-1} z^{k-1}.$$

Picking the k coefficients $s_0, s_1, \ldots, s_{k-1}$ is equivalent to selecting q_k, and, thus, $\mathbf{x}_k$. Notice that

$$\begin{aligned}\mathbf{x}_k = q_k(\mathbf{A})\mathbf{b} &= s_0\mathbf{b} + s_1\mathbf{A}\mathbf{b} + s_2\mathbf{A}^2\mathbf{b} + \cdots + s_{k-1}\mathbf{A}^{k-1}\mathbf{b} \\ &\in \operatorname{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}. \end{aligned} \tag{9.9}$$

In light of this characterization, we can express $\mathbf{x}_k$ as a linear combination of the vectors in this spanning set, or, equivalently, as a matrix-vector product involving the *Krylov matrix* $\mathbf{K}_k \in \mathbb{R}^{n\times k}$ and the vector $\mathbf{s} \in \mathbb{R}^k$ of coefficients:

The subspace in (9.9) is called a Krylov *subspace*, named after Russian academician Aleksey Krylov.

$$\mathbf{K}_k = \begin{bmatrix} \mathbf{b} & \mathbf{A}\mathbf{b} & \mathbf{A}^2\mathbf{b} & \cdots & \mathbf{A}^{k-1}\mathbf{b} \end{bmatrix} \in \mathbb{R}^{n\times k}, \qquad \mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{k-1} \end{bmatrix} \in \mathbb{R}^k.$$

Since $\mathbf{x}_k = \mathbf{K}_k\mathbf{s}$, we can write

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{b} - \mathbf{A}\mathbf{K}_k\mathbf{s},$$

and so minimizing $\|\mathbf{r}_k\|$ amounts to solving the least squares problem

$$\min_{\mathbf{s}\in\mathbb{R}^k} \|\mathbf{b} - (\mathbf{A}\mathbf{K}_k)\mathbf{s}\|, \tag{9.10}$$

As we will see in a moment, we always have $k \leq n$.

where $\mathbf{A}\mathbf{K}_k \in \mathbb{R}^{n\times k}$. By now you are experts in such least squares problems; after Chapter 7 can settle this optimization via the pseudoinverse:

$$\mathbf{s} = (\mathbf{A}\mathbf{K}_k)^+\mathbf{b}. \tag{9.11}$$

With this formula for $\mathbf{s}$, we can write down the iterate as

$$\mathbf{x}_k = \mathbf{K}_k\mathbf{s} \tag{9.12}$$

and the associated residual as

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{K}_k\mathbf{s}.$$

Before going any further, let us observe that at some iteration $k_\star \leq n$, *this procedure will produce the exact solution:*

$$\mathbf{x}_{k_\star} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b},$$

and so we only consider the case $k \leq n$. To see this, let $\lambda_1, \ldots, \lambda_n$ denote the eigenvalues of $\mathbf{A}$ (which are all nonzero, since $\mathbf{A}$ is invertible), and define the polynomial

$$p_n(z) = \big(1 - z/\lambda_1\big)\big(1 - z/\lambda_2\big)\cdots\big(1 - z/\lambda_n\big);$$

this polynomial has degree n and satisfies $p_n(0) = 1$. One can show that $p_n(\mathbf{A}) = \mathbf{0}$. Since

For example, if $\mathbf{A}$ is symmetric with eigendecomposition

$$\mathbf{A} = \sum_{j=1}^n \lambda_j \mathbf{v}_j\mathbf{v}_j^T,$$

then

$$\begin{aligned} p_n(\mathbf{A}) &= \sum_{j=1}^n p_n(\lambda_j)\mathbf{v}_j\mathbf{v}_j^T \\ &= \mathbf{0} \end{aligned}$$

since $p_n(\lambda_j) = 0$ for each eigenvalue λ_j of $\mathbf{A}$.

$$\|\mathbf{r}_n\| = \min_{\substack{\deg(p)\leq n \\ p(0)=1}} \|p(\mathbf{A})\mathbf{b}\| \leq \|p_n(\mathbf{A})\mathbf{b}\| = 0,$$

we see that $\mathbf{0} = \mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n$ and so $\mathbf{A}\mathbf{x}_n = \mathbf{b}$: so $\mathbf{x}_n$ is the exact solution. (It is possible that GMRES finds the exact solution sooner, at $k < n$ – this will happen, for example, if $\mathbf{A} = \mathbf{I}$ – but we are assured the GMRES must converge exactly by iteration $k = n$.)

To compute $\mathbf{x}_k$ using the formula (9.12) would require us to solve the least squares problem (9.10) involving the matrix $\mathbf{A}\mathbf{K}_k$ of dimension $n \times k$, requiring quite a bit of work at each step. Before we worry about this computational complexity, we will encounter even more profound stability problems with this approach.

Before exploring these problems, let us capture a few observations about the quality of this approximation.

Key properties of the GMRES Residuals

- The approximation subspace

  $$\mathcal{K}_k := \text{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}$$

  from which GMRES extracts the residual-minimizing iterate $\mathbf{x}_k \in \mathcal{K}_k$ expands as k increases, $\mathcal{K}_k \subseteq \mathcal{K}_{k+1}$, which implies that $\mathbf{x}_{k+1}$ is at least as good an approximation as was $\mathbf{x}_k$, implying

  $$\|\mathbf{r}_{k+1}\| \leq \|\mathbf{r}_k\|. \tag{9.13}$$

- Eventually k is increased enough so that $\mathcal{K}_{k+1} = \mathcal{K}_k$; this must happen at least when $k = n$ (since $\mathcal{K}_k \subseteq \mathbb{R}^n$, it cannot have dimension larger than n), if not before. At this stage, one can show that $\mathbf{x}_k$ equals the exact solution, with $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{0}$. Thus GMRES finds the exact solution in n steps or fewer. (In practice, we hope GMRES provides an accurate estimate for k much smaller than n.)

9.5.2 *A fatal flaw with this implementation*

The algorithm we have just proposed, summarized by the pseudoinverse formula

$$\mathbf{x}_k = (\mathbf{A}\mathbf{K}_k)^+\mathbf{b},$$

has a fatal flaw that makes it *entirely unsuitable* in practice. Suppose that $\mathbf{A} = \mathbf{A}^T$ has the eigendecomposition

$$\mathbf{A} = \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \mathbf{v}_j^T$$

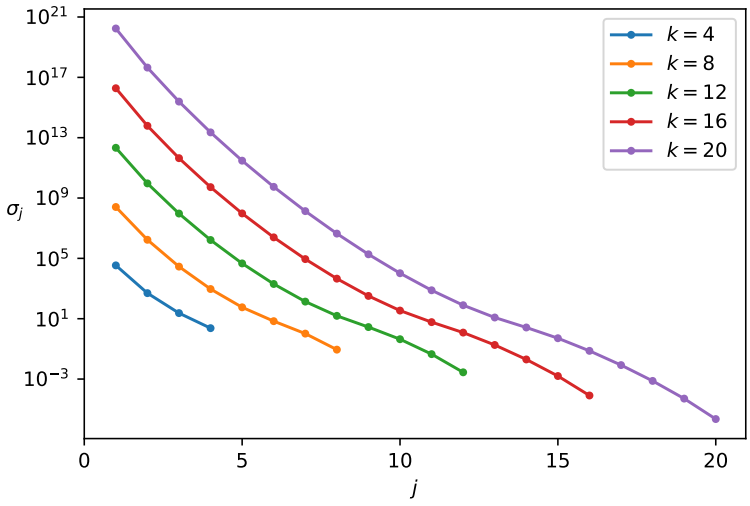


Figure 9.7: Singular values of the matrix $\mathbf{AK}_k$ at $k = 4, 8, 12, 16, 20$, for the matrix of dimension $n = 91$ in (9.14).

with eigenvalues ordered by decreasing magnitude

$$\lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n > 0.$$

Here we have stipulated that all the eigenvalues of $\mathbf{A}$ are positive and the largest of them is distinct, so $\lambda_1 > \lambda_j$ for $j = 2, \ldots, n$.

Now consider the columns of the matrix

$$\mathbf{AK}_k = \begin{bmatrix} \mathbf{Ab} & \mathbf{A}^2\mathbf{b} & \mathbf{A}^3\mathbf{b} & \cdots & \mathbf{A}^k\mathbf{b} \end{bmatrix} \in \mathbb{R}^{n \times k}.$$

We can express the pth column as a linear combination of the eigenvectors of $\mathbf{A}$,

$$\mathbf{A}^p\mathbf{b} = \sum_{j=1}^{n} \lambda_j^p (\mathbf{v}_j^T \mathbf{b}) \mathbf{v}_j.$$

Since λ_1 the largest magnitude eigenvalue, then (assuming $\mathbf{v}_1^T\mathbf{b} \neq 0$, the term

$$\lambda_1^p (\mathbf{v}_1^T \mathbf{b}) \mathbf{v}_1$$

will increasingly dominate the formula for $\mathbf{A}^p\mathbf{b}$:

the columns of $\mathbf{AK}_k$ *increasingly align with* $\mathbf{v}_1$ *as* k *increases*.

Indeed, the vectors $\mathbf{A}^p\mathbf{b}$ are (up to normalization) the iterates of the *power method*, a simple algorithm for computing the eigenvector associated with the largest magnitude eigenvalue.

Thus, the columns of $\mathbf{AK}_k$ gradually drift toward *linear dependence*: even if – strictly speaking – they remain linearly independent,

the singular values of $\mathbf{AK}_k$ *typically decay very rapidly.*

The extent of this decay can be startling. Consider the simple diagonal matrix

$$\mathbf{A} = \mathrm{diag}(1, 1.1, 1.2, \ldots, 9.9, 10) \in \mathbb{R}^{91 \times 91} \tag{9.14}$$

with right-hand side vector $\mathbf{b} = [1, 1, \ldots, 1]^T$. Figure 9.7 shows the singular values of $\mathbf{AK}_k$ for $K = 4, 8, 12, 16$, and 20. Consider the case

of $k = 12$: the singular values decay by roughly a factor of 10^{-15} from $\sigma_1(\mathbf{AK}_{12})$ to $\sigma_{12}(\mathbf{AK}_{12})$. Given our experience in the last two chapters, given such start singular value decay, we have little confidence that a solution $(\mathbf{AK}_k)^+\mathbf{b}$ will be computed with any accuracy.

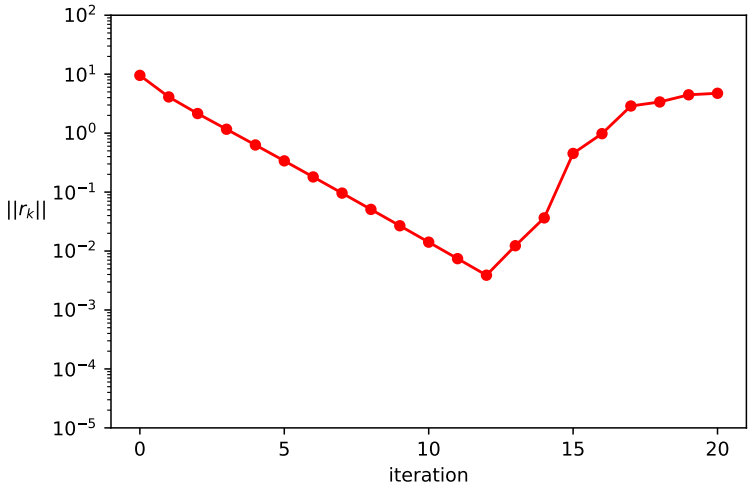


Figure 9.8: Convergence of GMRES, using the pseudoinverse formula $\mathbf{x}_k = (\mathbf{AK}_k)^+\mathbf{b}$ to compute the iterates, for the matrix $\mathbf{A}$ in (9.14). Numerical instability becomes obvious around step $k = 13$.

Does this singular value decay matter for the algorithm? Yes, indeed! Figure 9.8 shows that GMRES initially converges steadily up to that $k = 12$ iteration. At that point, we see a noticeable *increase* in the norm of the computed residual, violating the theoretical property (9.13) that insists that convergence be monotone decreasing:

$$\|\mathbf{r}_{k+1}\| \leq \|\mathbf{r}_k\|.$$

If this implementation of the minimum residual method fails for such a simple matrix $\mathbf{A}$, we can have no hope it will be reliable for the kinds of large-scale linear systems to which we hope to apply this algorithm. A better approach is needed.

9.5.3 *A clever fix: the GMRES implementation*

By this stage in the course your instinct by to "fix" the formula $\mathbf{x}_k = (\mathbf{AK}_k)^+\mathbf{b}$ by applying singular value truncation or Tikhonov regularization. In this case, there is a better approach:

reformulate the problem to avoid the instability.

For the image deblurring examples considered in the previous chapters, the ill-posedness was fundamental to the blurring operation: this blurring naturally maps distinct crisp images close together.

In our present setting, the singular value decay in $\mathbf{AK}_k$ does not capture a fundamental aspect of the minimum residual calculation;

rather, it simply reflects the fact that *we chose a poor basis* for the space $\mathcal{R}(\mathbf{K}_k)$. A more effective implementation of GMRES follows from using a better basis for this subspace.

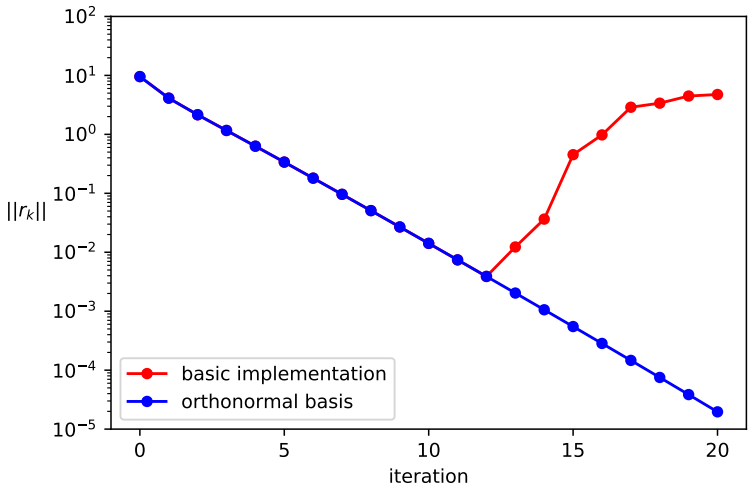


Figure 9.9: Convergence of GMRES, using the improved approach using an orthonormal basis for the subspace $\mathcal{K}_k$, for the matrix $\mathbf{A}$ in (9.14). The numerical stability issue that occurs with the pseudoinverse solution is avoided.

9.5.4 *An acceleration for symmetric* $\mathbf{A}$

9.5.5 *An compromise for nonsymmetric* $\mathbf{A}$*: restarting*

9.5.6 *Convergence theory*

$$\mathbf{A} = \text{diag}(1, 2, 3, \ldots, 19, 20) \in \mathbb{R}^{20\times 20}. \tag{9.15}$$

9.6 *Kaczmarz's Method*

9.7 *Alternative iterations for nonsymmetric* $\mathbf{A}$

Bi-Conjugate Gradients (BiCG), Conjugate Gradients Squared (CGS), Quasi-Minimum Residual (QMR), Transpose-Free QMR (TFQMR), Bi-Conjugate Gradient Stabilized (BiCGSTAB), Induced Dimension Reduction (IDR) methods

9.8 *Preconditioning*

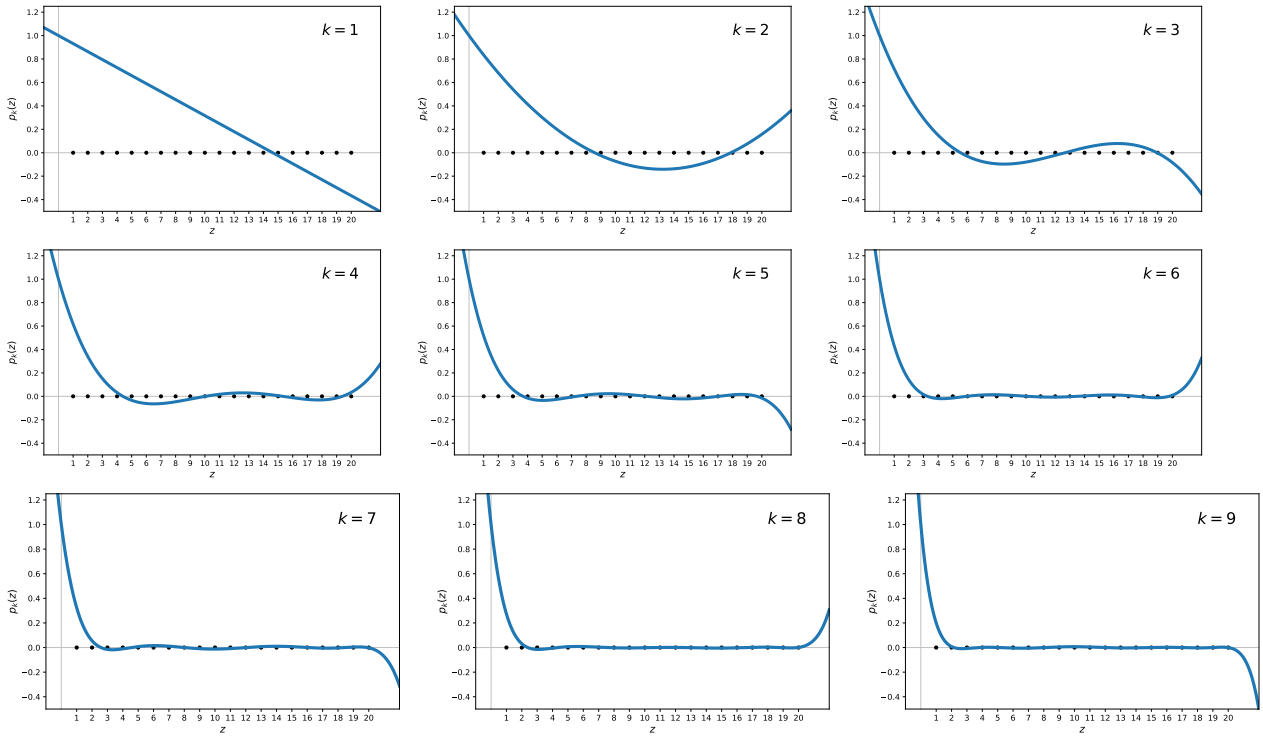


Figure 9.10: GMRES residual polynomials p_k for the matrix $\mathbf{A}$ in (9.15) with a random $\mathbf{b}$ vector.

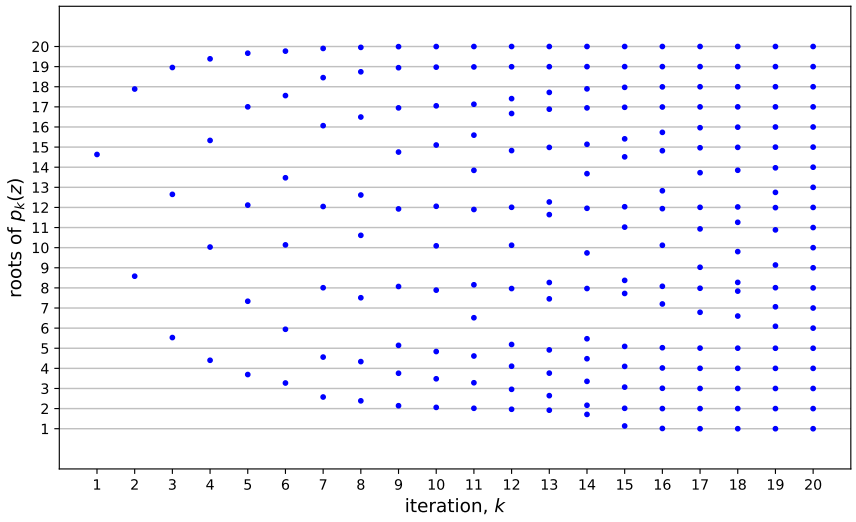
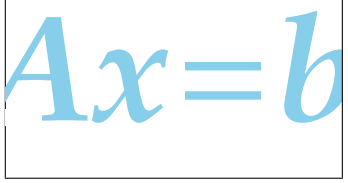


Figure 9.11: Roots of the GMRES residual polynomial p_k for $k = 1, 2, \ldots, 20$. (For each value of k, the roots are displayed on a vertical line. The gray horizontal lines show the eigenvalues of $\mathbf{A}$.) Notice how the roots "find" the eigenvalues of $\mathbf{A}$, though we have not given GMRES any information about these eigenvalues.

Matrix Methods for Computational Modeling and Data Analytics

CMDA Program · Virginia Tech

Mark Embree
embree@vt.edu



version of 12 June 2023

Chapter 10
Iterative methods for nonlinear optimization

THUS FAR WE HAVE FOCUSED ON LINEAR PROBLEMS, but many modeling problems give rise to another vital problem: minimizing a nonlinear function. In the final component of these notes we give a brief introduction to this expansive area.

10.1 Overview of nonlinear optimization

The field of optimization spans a tremendous variety of problems, with specialized algorithms tailored to exploit the special properties of each class. For example, the field includes optimization of

- Smooth functions $f : \mathbb{R}^n \to \mathbb{R}$ with no constraints;
- Smooth functions $f : \Omega \to \mathbb{R}$ with constraints limiting $\Omega \subset \mathbb{R}$;
- Functions $f : \mathbb{R}^n \to \mathbb{R}$ such as

$$f(\mathbf{x}) = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 + \lambda \|\mathbf{x}\|_1, \qquad \|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n|$$

  which does not have a derivative to some points $\mathbf{x} \in \mathbb{R}$;
- Linear functions $\mathbf{c}^T\mathbf{x}$ subject to equality and inequality constraints;
- Functions f where some variables can only take integer values.

In these notes, we shall focus on the first class of problems: minimization of smooth, multivariable functions in the absence of constraints. These notes were inspired by the important and highly recommended textbook *Nonlinear Optimization* by JORGE NOCEDAL and STEPHEN J. WRIGHT, which addresses many more details, and covers a far broader range of optimization problems.

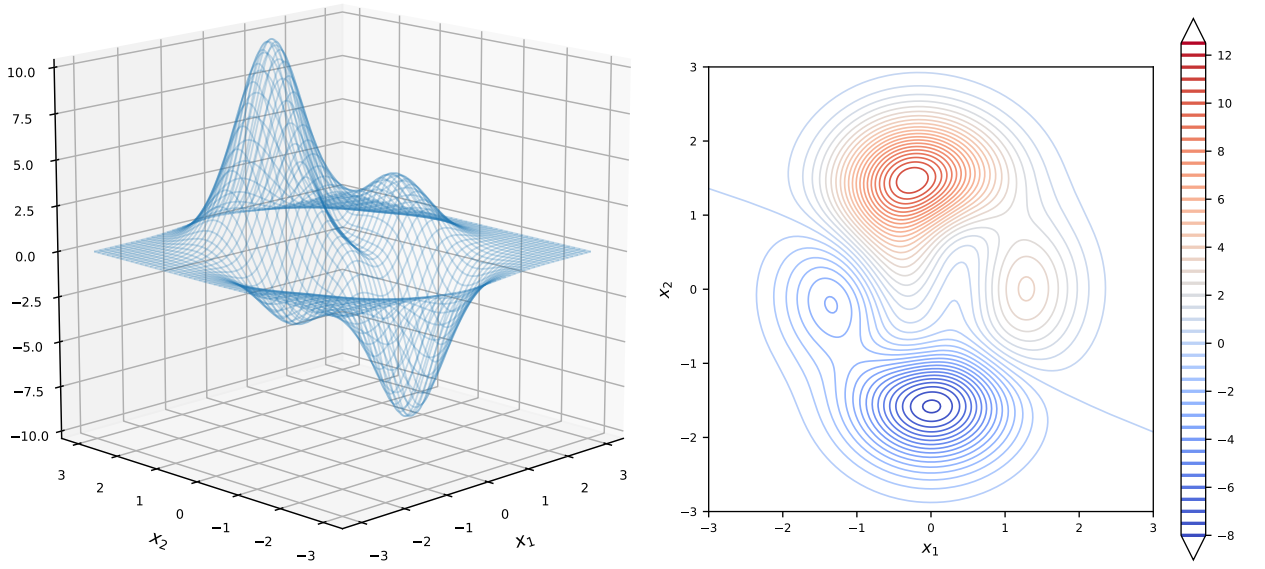


Figure 10.1: An example of a nonlinear function of two variables, $f : \mathbb{R}^2 \to \mathbb{R}$ (the `peaks` function in MATLAB), illustrated by a wireframe plot on the left and a contour plot on the right.

10.2 *Functions of several variables*

Figure 10.1 shows a function $f : \mathbb{R}^2 \to \mathbb{R}$ that we will use throughout this chapter to illustrate basic concepts from nonlinear optimization. This figure demonstrates two ways to visualize a function of two variables: on the left, we think of the variables x_1 and x_2 as coordinates on a map, the value of $f(\mathbf{x})$ at $\mathbf{x} = [x_1, x_2]^T$ as the height of the surface. While such views are only possible with $n = 2$ variables, they inform the language with which we describe optimization algorithms, e.g., going "downhill" toward the local minimum, or getting caught on a "saddle point." The plot on the right of Figure 10.1 is a contour plot, showing lines where $f(\mathbf{x})$ takes on a constant value that can be read off the colorbar.

10.2.1 *Taylor expansion in higher dimensions*

For a smooth function $f : \mathbb{R} \to \mathbb{R}$ of one variable, the Taylor expansion

$$f(x+p) = f(x) + pf'(x) + \frac{p^2}{2}f''(x) + \cdots$$

describes the behavior of f near the point x, i.e., for small $|p|$. Truncating the Taylor series after the first three terms gives

$$\phi(x+p) = f(x) + f'(x)p + \frac{1}{2}f''(x)p^2,$$

a quadratic polynomial in the variable p that *interpolates* f, f', and f'' at x, meaning that ϕ and its first two derivatives match those values of f at x:

$$\phi(x) = f(x), \quad \phi'(x) = f'(x), \quad \phi''(x) = f''(x).$$

Thus, the truncated Taylor series gives a *polynomial model* that matches the behavior of f locally around the single point $x \in \mathbb{R}$. The quality of the model likely degrades as $x + p$ gets farther from x.

A perfect generalization of the Taylor series exists, for functions $f : \mathbb{R}^n \to \mathbb{R}$ of more variables. Naturally such an expansion will require a multivariable generalization of the derivatives $f'(x)$ and $f''(x)$, called the *gradient* and Hessian of f.

Definition 10.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable at $\mathbf{x} \in \mathbb{R}^n$. Then the* gradient *of f at $\mathbf{x}$ is the vector of first partial derivatives,*

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^n.$$

If all second partial derivatives of f exist at $\mathbf{x}$, the Hessian *of f at $\mathbf{x}$ is the matrix containing all second partial derivatives,*

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_n \partial x_1} f(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_1 \partial x_n} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_n^2} f(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

The (j, k) entry of the Hessian is

$$\left(\nabla^2 f(\mathbf{x})\right)_{j,k} = \frac{\partial}{\partial x_k} \nabla f(\mathbf{x}) = \frac{\partial^2}{\partial x_k \partial x_j} f(\mathbf{x}).$$

If all these second derivatives are continuous at $\mathbf{x}$, the mixed partial derivatives match,

$$\frac{\partial^2}{\partial x_k \partial x_j} f(\mathbf{x}) = \frac{\partial^2}{\partial x_j \partial x_k} f(\mathbf{x}),$$

and hence the Hessian is a symmetric matrix.

With these definitions in place, we can state the generalization of the Taylor series to the multivariable setting.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be smooth function at $\mathbf{x} \in \mathbb{R}^n$. For $\mathbf{p} \in \mathbb{R}^n$ with sufficiently small $\|\mathbf{p}\|$, we have the Taylor approximation

$$f(\mathbf{x} + \mathbf{p}) = f(\mathbf{x}) + \mathbf{p}^T \nabla f(\mathbf{x}) + \frac{1}{2} \mathbf{p}^T \nabla^2 f(\mathbf{x}) \mathbf{p} + \mathcal{O}(\|\mathbf{p}\|^3). \tag{10.1}$$

For our purpose it suffices to only consider the Taylor expansion up through the quadratic term involving the Hessian. The next term in the approximation, incorporating the third derivatives, could be expressed using an $n \times n \times n$ *tensor* of mixed third partial derivatives.

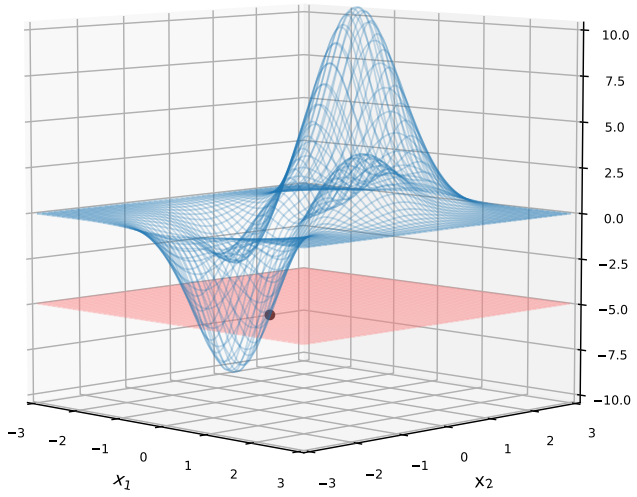


Figure 10.2: The function from Figure 10.1 (blue), with its constant-order approximation (red) that interpolates f at the point $\mathbf{x} = [0.5, -1.25]^T$ (black dot).

TAYLOR series might seem obscure at first encounter, but they are a fundamental tool of mathematical modeling. To help build your intuition for the TAYLOR series, we will show examples of the TAYLOR approximations of increasing quality. The most basic approximation comes from the constant-order term,

$$f(\mathbf{x} + \mathbf{p}) \approx f(\mathbf{x}).$$

Here we regard $\mathbf{x} \in \mathbb{R}^n$ as a fixed vector and $\mathbf{p} \in \mathbb{R}^n$ as the variable. The function

$$\phi(\mathbf{x} + \mathbf{p}) = f(\mathbf{x})$$

is just a constant that passes through the $f(\mathbf{x} + \mathbf{p})$ surface when $\mathbf{p} = 0$. Figure 10.2 shows this approximation $\phi(\mathbf{x} + \mathbf{p}) = f(\mathbf{x})$ as a red plane passing through f at $\mathbf{x} = [0.5, -1.25]^T$. This red plane only gives an accurate approximation for very small $\|\mathbf{p}\|$; it entirely misses f's prominent slope.

To capture that slope, add the first-order term into the approximation, now defining

$$\phi(\mathbf{x} + \mathbf{p}) = f(\mathbf{x}) + \mathbf{p}^T \nabla f(\mathbf{x}).$$

The function ϕ now traces out a plane whose slope (gradient) matches that slope f at $\mathbf{x}$:

$$\phi(\mathbf{x}) = f(\mathbf{x}), \qquad \nabla\phi(\mathbf{x}) = \nabla f(\mathbf{x}).$$

Figure 10.3 shows this improved approximation, with the desired slope matching at $\mathbf{x} = [0.5, -1.25]^T$.

On can obtain a more quantitative impression of this approximation by comparing contour plots for f and ϕ. Figure 10.4 shows

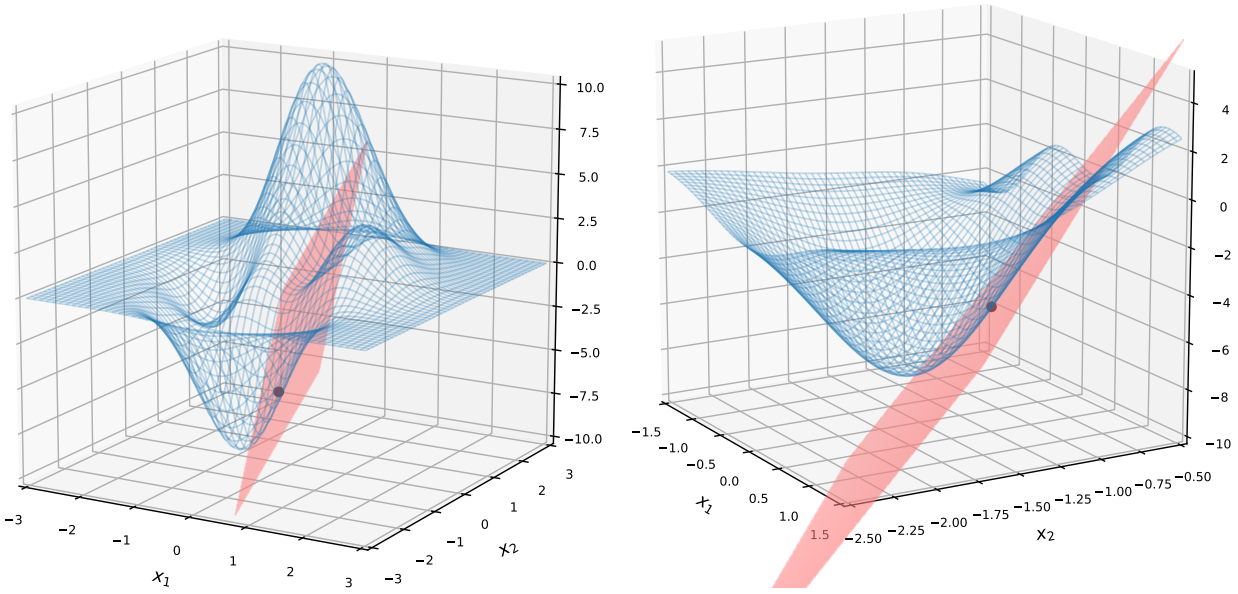


Figure 10.3: The function from Figure 10.1 (blue), with its first-order approximation at the point $\mathbf{x} = [0.5, -1.25]^T$ (red). The plot on the right zooms in a bit closer near $\mathbf{x}$.

lines of constant $f(\mathbf{x})$ values in black, and with $\phi(\mathbf{x})$ at the same values shown in red. (Since ϕ describes a plane, its contours are straight lines.) In the immediate vicinity of the expansion point $\mathbf{x} = [0.5, -1.25]^T$ (black dot), the red and black contours align nicely. Before long, this approximation degrades, and the black contours of f fall away from the red contours of ϕ.

To get a better approximation still, extend ϕ through the addition

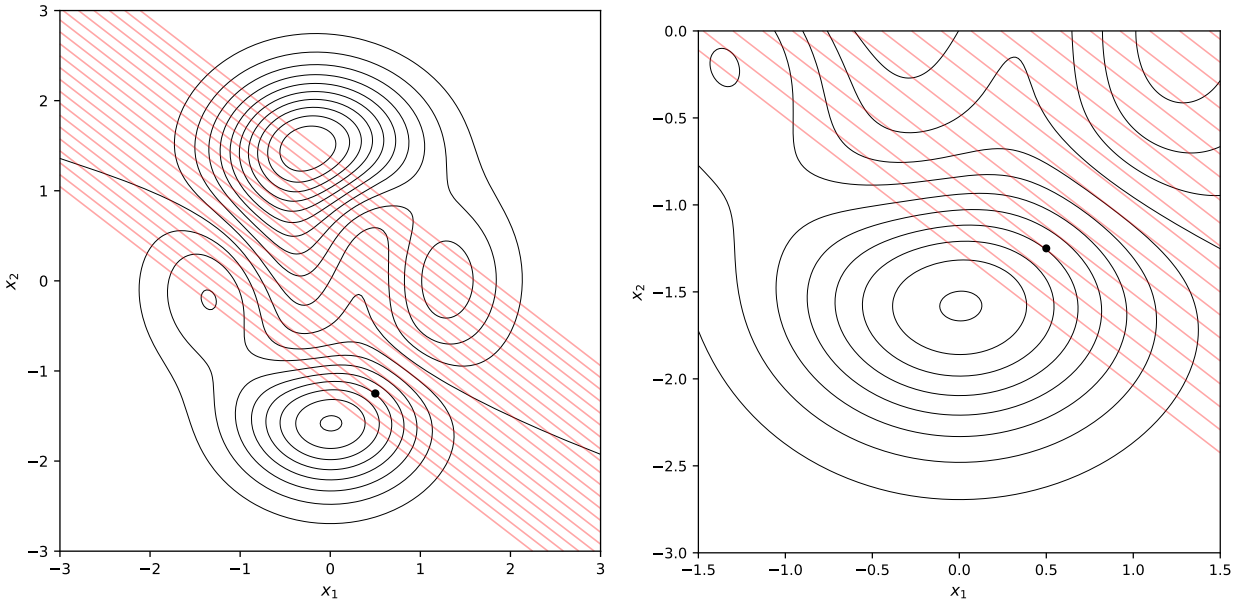


Figure 10.4: Contour lines of the function from Figure 10.1 (black), contrasted with the same contour lines for the first-order Taylor approximation at $\mathbf{x} = [0.5, -1.25]^T$ (red). The plot on the right zooms in around $\mathbf{x}$.

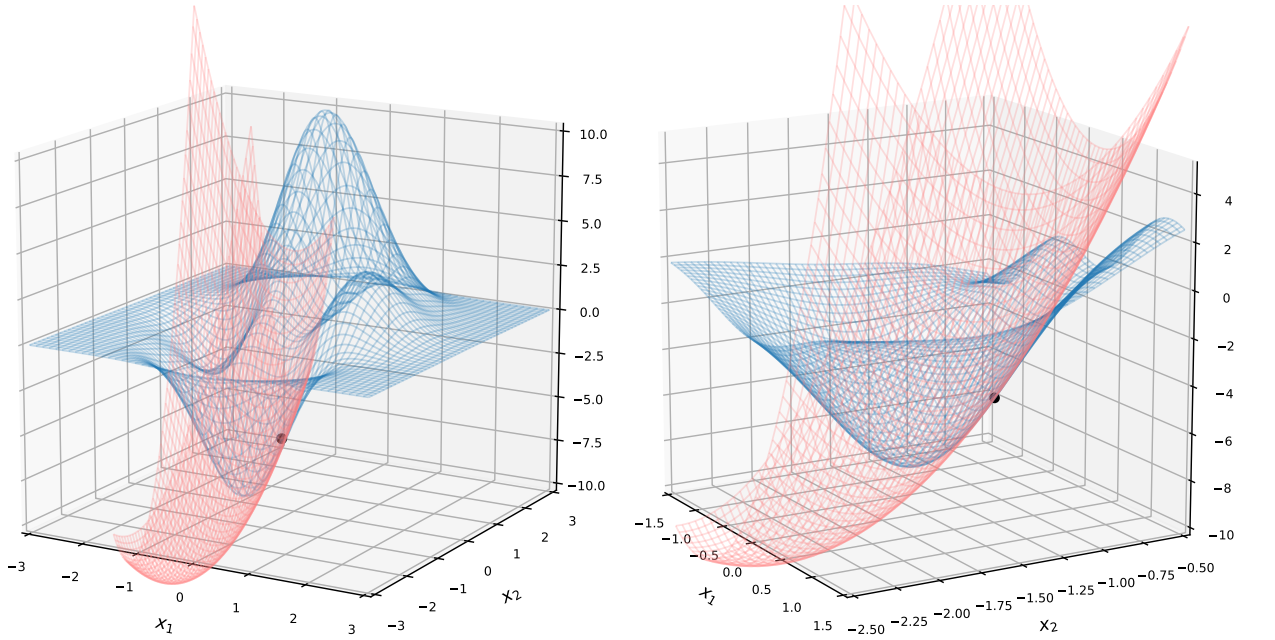


Figure 10.5: The function from Figure 10.1 (blue), with its second-order approximation at the point $\mathbf{x} = [0.5, -1.25]^T$ (red). The plot on the right zooms in a bit closer near $\mathbf{x}$.

of the quadratic term,

$$\phi(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \mathbf{p}^T\nabla f(\mathbf{x}) + \frac{1}{2}\mathbf{p}^T\nabla^2 f(\mathbf{x})\mathbf{p}.$$

The graph of ϕ will be a paraboloid (opening up or down), or hyperbolic paraboloid (a saddle surface), depending on properties of the Hessian described in Section 10.2.2. The surface will align with f near $\mathbf{x}$, matching f and its first two derivatives at $\mathbf{x}$:

$$\phi(\mathbf{x}) = f(\mathbf{x}), \qquad \nabla\phi(\mathbf{x}) = \nabla f(\mathbf{x}), \qquad \nabla^2\phi(\mathbf{x}) = \nabla^2 f(\mathbf{x}).$$

Figure 10.5 gives two views of this quadratic approximation. These plots immediately suggest an algorithm for minimizing f: Newton's method approximates the minimum of f with the minimum of its quadratic approximation about some point $\mathbf{x}_k$, i.e., the minimum of the red surface. This new point, called $\mathbf{x}_{k+1}$, will not typically be the minimum of f, but under favorable conditions it will be closer to that minimum than was $\mathbf{x}_k$. A quadratic approximation to f about $\mathbf{x}_{k+1}$ will provide an even better approximation.

Figure 10.6 compares contour plots of f and ϕ. Compare these plots to the linear approximations in Figure 10.4. Near the black dot at $\mathbf{x} = [0.5, -1.25]^T$, the red contours bend with similar curvature as the contours of f, and the contours have very similar spacing (reflecting the equal slope of the two surfaces). The quadratic model is quite accurate near $\mathbf{x}$, but away from $\mathbf{x}$ drifts away, attaining a lower minimum value than f does. Moving $\mathbf{x}$ closer to the point

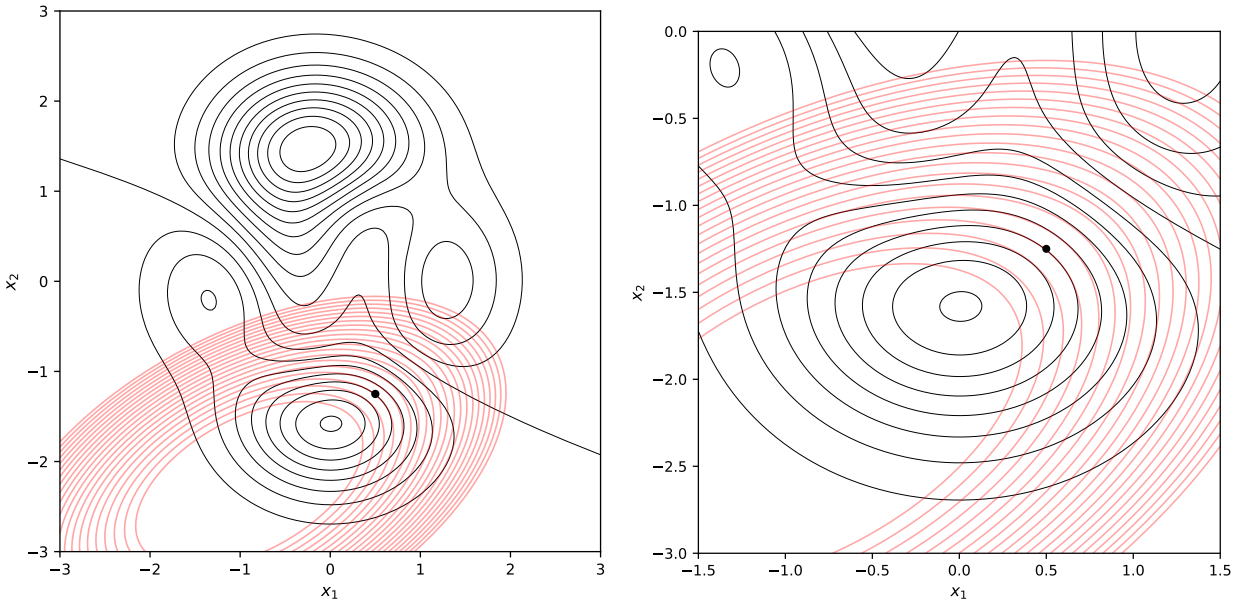


Figure 10.6: Contour lines of the function from Figure 10.1 (black), contrasted with the same contour lines for the second-order Taylor approximation at $\mathbf{x} = [0.5, -1.25]^T$ (red). The plot on the right zooms in around $\mathbf{x}$. Note how closely the contours match in the vicinity of the expansion point $\mathbf{x}$.

where f attains its minimum, we would hope and expect the Taylor expansion to better capture the characteristics of f at its minimum.

As we shall see in the sections ahead, these Taylor approximations provide the key insights into f that guide the design of numerical methods for minimizing f.

10.2.2 *Critical points and the behavior of f nearby*

The Taylor expansion (10.1 deeply informs our task of minimizing $f : \mathbb{R}^n \to \mathbb{R}$, identifying properties that must be satisfied at a local minimum point. Look back at the first-order approximations in Figure 10.3. The red plane $f(\mathbf{x}) + \mathbf{p}^T \nabla f(\mathbf{x})$ has the same slope as f near $\mathbf{x}$, as encoded in the gradient vector

$$\nabla f(\mathbf{x}).$$

If this vector is nonzero, there is some direction at which

$$\mathbf{p}^T \nabla f(\mathbf{x}) < 0$$

and hence f cannot obtain a local minimum at such an $\mathbf{x}$. The only way that $\mathbf{x}$ can *potentially* be a local minimum is when the gradient is the zero vector, $\nabla f(\mathbf{x}) = \mathbf{0}$.

First Order Necessary Conditions for Optimality
Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable at $\mathbf{x} \in \mathbb{R}^n$.
If f attains a local minimum at $\mathbf{x}$, then

$$\nabla f(\mathbf{x}) = \mathbf{0}.$$

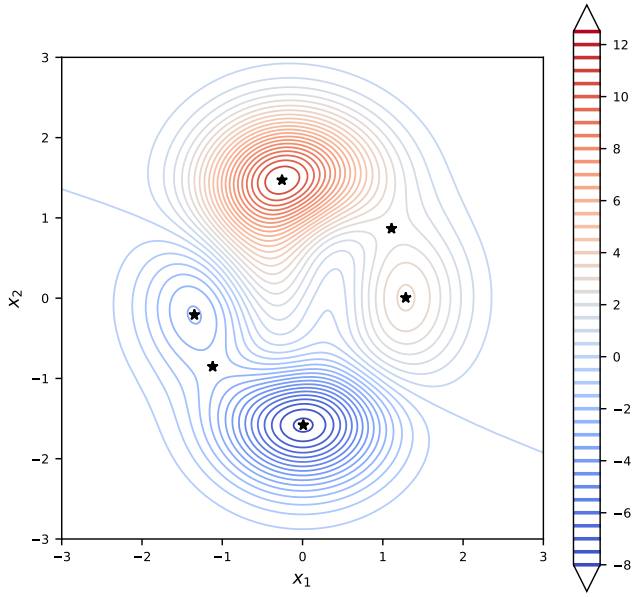


Figure 10.7: Six critical points (marked by stars) for the function illustrated in Figure 10.1. The critical points correspond to two local minima, two saddle points, and two local maxima.

This key property, which must hold in order for $\mathbf{x}$ to be a minimizer of f, clarifies our algorithmic task: we seek points $\mathbf{x} \in \mathbb{R}^n$ where $\nabla f(\mathbf{x}) = \mathbf{0}$.

Definition 10.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable at $\mathbf{x}_\star \in \mathbb{R}^n$. We say that $\mathbf{x}_\star$ is a* critical point *of f if the gradient of f at $\mathbf{x}_\star$ is zero,*

$$\nabla f(\mathbf{x}_\star) = \mathbf{0}.$$

Figure 10.7 identifies six critical points for the function shown in Figure 10.1. Notice that only two of these critical points are local minimizers: two are local *maximizers*, and two others are *saddle points* where f increases in one direction but decreases in another. Clearly being a critical point is only a *necessary* condition for optimality; it is not *sufficient*. No surprise, a much deeper understanding can be teased out from the next term in the Taylor series.

10.2.3 Learning from second order information

At a critical point $\mathbf{x}_\star \in \mathbb{R}^n$, f has zero gradient, and so the variation of the function is largely controlled by the Hessian matrix. From (10.1) we have

$$f(\mathbf{x}_\star + \mathbf{p}) = f(\mathbf{x}_\star) + \frac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x}_\star)\mathbf{p} + \mathcal{O}(\|\mathbf{p}\|^3).$$

To simplify the notation a bit, denote the Hessian at $\mathbf{x}_\star$ by

$$\mathbf{H} := \nabla^2 f(\mathbf{x}_\star).$$

For smooth f the Hessian is symmetric, and so it has the eigendecomposition

$$\mathbf{H} = \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \mathbf{v}_j^T$$

as detailed in Chapter 4. Then for any $\mathbf{p} \in \mathbb{R}^n$,

$$\mathbf{p}^T \mathbf{H} \mathbf{p} = \sum_{j=1}^{n} \lambda_j (\mathbf{p}^T \mathbf{v}_j)(\mathbf{v}_j^T \mathbf{p}) = \sum_{j=1}^{n} \lambda_j (\mathbf{v}_j^T \mathbf{p})^2. \tag{10.2}$$

Take a moment to savor this equation, which unlocks our understanding of multivariable functions in the proximity of a critical point.

Notice that $(\mathbf{v}_j^T \mathbf{p})^2 \geq 0$, and so the *sign* of the contribution of that the jth term $\lambda_j (\mathbf{v}_j^T \mathbf{p})^2$ makes to $\mathbf{p}^T \mathbf{H} \mathbf{p}$ is controlled entirely by λ_j. Returning to the Taylor expansion,

$$f(\mathbf{x}_\star + \mathbf{p}) = f(\mathbf{x}_\star) + \frac{1}{2} \left(\sum_{j=1}^{n} \lambda_j (\mathbf{v}_j^T \mathbf{p})^2 \right) + \mathcal{O}(\|\mathbf{p}\|^3),$$

we see that for sufficiently small $\|\mathbf{p}\|$, the value of $f(\mathbf{x}_\star + \mathbf{p})$ is dominated by the Hessian term, whose eigenvalues thus control the local increase or decrease.

- If all eigenvalues of $\mathbf{H}$ are *positive*, $\lambda_1, \ldots, \lambda_n > 0$, then for any nonzero $\mathbf{p} \in \mathbb{R}^n$

  $$\mathbf{p}^T \mathbf{H} \mathbf{p} = \sum_{j=1}^{n} \lambda_j (\mathbf{v}_j^T \mathbf{p})^2 > 0$$

  and so small changes $\mathbf{p}$ to $\mathbf{x}_\star$ *increase* the objective function,

  $$f(\mathbf{x}_\star + \mathbf{p}) > f(\mathbf{x}_\star),$$

  and so $\mathbf{x}_\star$ is a *local minimum*.

When all eigenvalues of $\mathbf{H} = \mathbf{H}^T$ are positive, then $\mathbf{H}$ is *positive definite*; equivalently, $\mathbf{p}^T \mathbf{H} \mathbf{p} > 0$ for all $\mathbf{p} \neq \mathbf{0}$.

- If all eigenvalues of $\mathbf{H}$ are *negative*, $\lambda_1, \ldots, \lambda_n < 0$, then for any nonzero $\mathbf{p} \in \mathbb{R}^n$,

  $$\mathbf{p}^T \mathbf{H} \mathbf{p} = \sum_{j=1}^{n} \lambda_j (\mathbf{v}_j^T \mathbf{p})^2 < 0$$

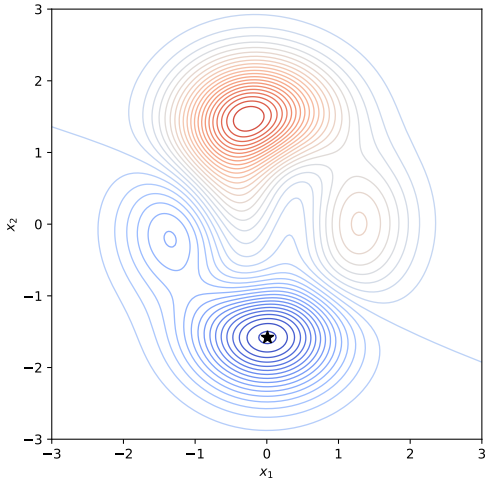
  and so small changes $\mathbf{p}$ to $\mathbf{x}_\star$ *decrease* the objective function,

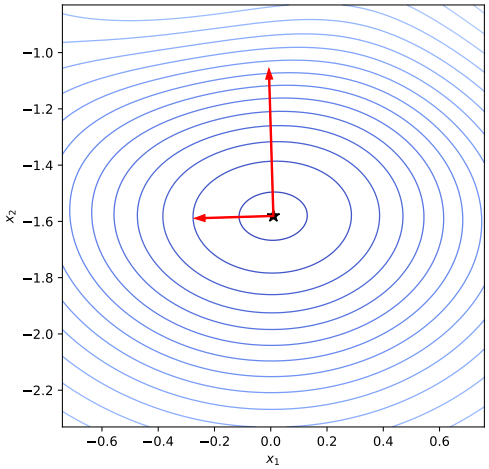
  $$f(\mathbf{x}_\star + \mathbf{p}) < f(\mathbf{x}_\star),$$

  and so $\mathbf{x}_\star$ is a *local maximum*.

When all eigenvalues of $\mathbf{H} = \mathbf{H}^T$ are negative, we say $\mathbf{H}$ is *negative definite*.

- What if $\mathbf{H}$ has a mix of positive and negative eigenvalues? Then the sign of $\mathbf{p}^T \mathbf{H} \mathbf{p}$ will depend on $\mathbf{p}$: some choices for $\mathbf{p}$ will give $f(\mathbf{x}_\star + \mathbf{p}) > f(\mathbf{x}_\star)$ while others will give $f(\mathbf{x}_\star + \mathbf{p}) < f(\mathbf{x}_\star)$. In such cases $\mathbf{x}_\star$ is neither a local minimum nor maximum; we call it a *saddle point*.

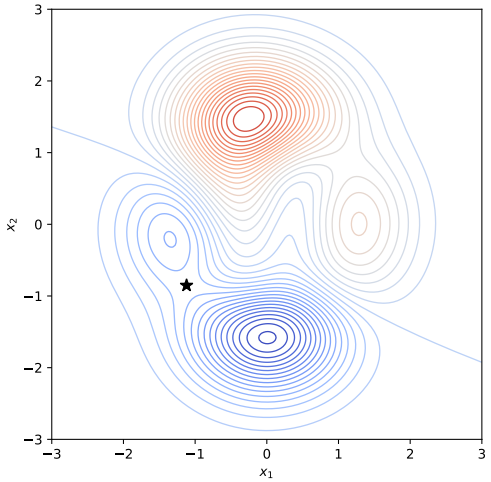


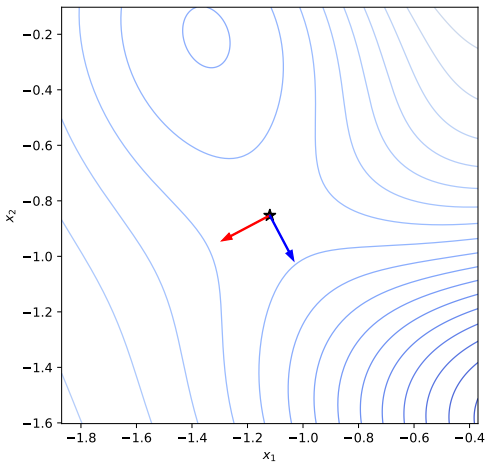


local minimum

$\lambda_1 = 32.4713$

$\lambda_2 = 16.2393$

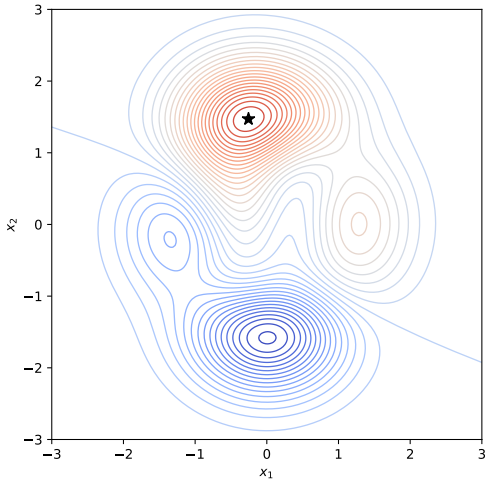


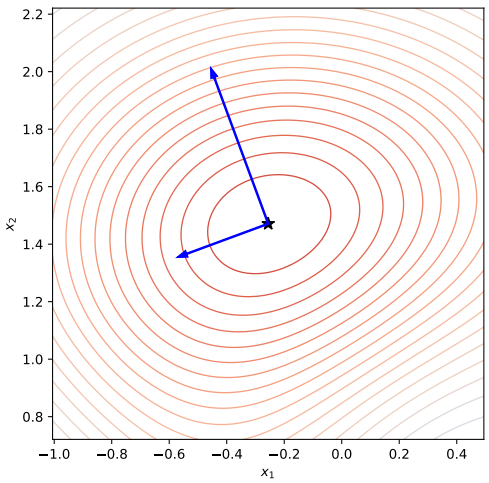


saddle point

$\lambda_1 = 10.6549$

$\lambda_2 = -9.8849$





local maximum

$\lambda_1 = -36.1728$

$\lambda_2 = -20.1383$

Figure 10.8: The arrows show eigenvectors of the Hessian at three critical points: a local minimum (top: two positive eigenvalues), a saddle point (middle: one positive and one negative eigenvalue), and a local maximum (bottom: two negative eigenvalues).

We can squeeze a bit more insight out of the Hessian. Suppose we nudge away from $\mathbf{x}_\star$ in the direction of the kth eigenvector of $\mathbf{H}$, i.e., for some small $\varepsilon > 0$ choose

$$\mathbf{p} = \varepsilon \mathbf{v}_k.$$

Then by the orthonormality of the eigenvectors of the symmetric matrix $\mathbf{H}$,

$$\mathbf{p}^T \mathbf{H} \mathbf{p} = \sum_{j=1}^{n} \lambda_j \varepsilon^2 (\mathbf{v}_j^T \mathbf{v}_k)^2 = \varepsilon^2 \lambda_k,$$

and so

$$f(\mathbf{x}_\star + \varepsilon \mathbf{v}_k) = f(\mathbf{x}_\star) + \lambda_k \varepsilon^2 + \mathcal{O}(\varepsilon^3).$$

This formula also hints at the complications that arise when the Hessian has a *zero* eigenvalue: then one must look inside the $\mathcal{O}(\varepsilon^3)$ term (the *third order* term in the Taylor series) to see how $f(\mathbf{x}_\star + \varepsilon \mathbf{v}_k)$ behaves.

This formula means

the eigenvalue λ_k describes how *fast or slow* $f(\mathbf{x}_\star + \mathbf{p})$ is changing in the $\mathbf{p} = \varepsilon \mathbf{v}_k$ direction.

The larger $|\lambda_k|$ is, the faster $f(\mathbf{x}_\star + \varepsilon \mathbf{v}_k)$ is changing, and hence the *tighter* the contour lines will be around $\mathbf{x}_\star$ in the $\mathbf{v}_k$ direction. Figure 10.8 illustrates this behavior for a local minimum, a local maximum, and a saddle point. In these plots, the red eigenvectors point in directions of increase ("uphill") while the blue vectors point in directions of decrease ("downhill"). The length of the vector corresponds to $|\lambda_k|$: the longer the vector, the tighter the contour lines.

10.3 Line search methods

Having studied properties of nonlinear functions, we are ready to go looking for a critical point. Once a critical point has been found, we could verify its minimality by computing the Hessian and checking if its eigenvalues are positive.

We shall focus on a class of algorithms known as *line search methods*. These methods seek a minimum of $f : \mathbb{R}^n \to \mathbb{R}$, starting from an initial guess $\mathbf{x}_0$, via iterations of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + c_k \mathbf{p}_k.$$

The vector $\mathbf{p}_k \in \mathbb{R}^n$, called the *search direction*, is typically determined by local properties of the function f around $\mathbf{x}_k$ (e.g., the gradient $\nabla f(\mathbf{x}_k)$). The constant c_k, the *step length*, controls how far the iterate $\mathbf{x}_{k+1}$ steps in the $\mathbf{p}_k$ direction.

The kth iteration of a line search method has two stages:

- determine the search direction $\mathbf{p}_k \in \mathbb{R}^n$;
- determine the step length $c_k \in \mathbb{R}$, i.e., how far to step in the $\mathbf{p}_k$ direction.

10.3.1 Search direction selection

What makes for a good search direction $\mathbf{p}_k$? How can we determine a good direction without too much computational effort? A reasonable minimum requirement is that $\mathbf{p}_k$ should point downhill.

Definition 10.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable at $\mathbf{x}_k \in \mathbb{R}^n$. We say that $\mathbf{p} \in \mathbb{R}^n$ is a* descent direction *for f at $\mathbf{x}_k$ provided*

$$\mathbf{p}^T \nabla f(\mathbf{x}_k) < 0,$$

which amounts to

$$\cos \angle(\mathbf{p}, \nabla f(\mathbf{x}_k)) \in [-1, 0),$$

and thus

$$\angle(\mathbf{p}, \nabla f(\mathbf{x}_k)) \in (-3\pi/2, -\pi/2),$$

Consider all descent directions $\mathbf{p}$ of unit norm, $\|\mathbf{p}\| = 1$, and suppose we make a step of size $\varepsilon\mathbf{p}$ for very small $\varepsilon > 0$. By the Cauchy–Schwarz inequality (Theorem 2.3),

$$|(\varepsilon\mathbf{p})^T \nabla f(\mathbf{x}_k)| \leq \|\varepsilon\mathbf{p}\| \|\nabla f(\mathbf{x}_k)\| = \varepsilon \|\nabla f(\mathbf{x}_k)\|.$$

For the special direction

$$\mathbf{p} = -\frac{\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|},$$

this inequality is sharp:

$$|(\varepsilon\mathbf{p})^T \nabla f(\mathbf{x}_k)| = \varepsilon \frac{\|(\nabla f(\mathbf{x}_k))^T \nabla f(\mathbf{x}_k)\|}{\|\nabla f(\mathbf{x}_k)\|} = \varepsilon \|\nabla f(\mathbf{x}_k)\|.$$

Combining these observations with the Taylor series

$$f(\mathbf{x}_k + \varepsilon\mathbf{p}) = f(\mathbf{x}_k) + (\varepsilon\mathbf{p})^T \nabla f(\mathbf{x}_k) + \mathcal{O}(\varepsilon)^2,$$

reveals that of all the choices we can make for the unit vector $\mathbf{p}$, the one that gives the most rapid decrease for small $\varepsilon \to 0$ (so the linear term dominates the $\mathcal{O}(\varepsilon^2)$ terms) is the *steepest descent direction* that points opposite the gradient. We typically remove the normalization factor for notational convenience.

Definition 10.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable at $\mathbf{x}_k \in \mathbb{R}^n$. The* steepest descent direction *for f at $\mathbf{x}_k$ is the vector*

$$\mathbf{p} = -\nabla f(\mathbf{x}_k).$$

The steepest descent direction looks like the best choice of $\mathbf{p}_k$ for a given value of $\mathbf{x}_k$, but there are a few reasons why we might prefer other descent directions that are not locally optimal but eventually lead to faster convergence.

- The gradient might be expensive to compute for complicated f or large n, causing us to favor directions $\mathbf{p}_k$ that are faster to compute.

- For some problems, line search algorithms based on the steepest descent direction can get stuck in a rut, repeating search directions as the method steps down a narrow valley. Methods (like the *conjugate gradient* algorithm) that force greater variety in the search directions can lead to faster convergence.

10.3.2 Step-length selection

Once the search direction $\mathbf{p}_k$ is selected, how should one choose the step-length c_k so the line-search method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + c_k \mathbf{p}_k$$

converges as quickly as possible?

One might naturally think of defining a function

$$g(c) := f(\mathbf{x}_k + c\mathbf{p}_k) \tag{10.3}$$

and minimizing g over all choices of $c > 0$. However, in many cases this would involve optimizing a nonlinear function of the single variable $c \in \mathbb{R}$; to evaluate $g(c)$ one must evaluate the n-variable function $f(\mathbf{x}_k + c\mathbf{p}_k)$, so finding this optimal c could be quite expensive. Typically we settle for heuristics, which can be crude as setting c to a constant value (e.g., the "learning rate" in machine learning optimization), to the more elaborate techniques we next discuss.

One must balance two competing goals:

- We want c_k to be sufficiently small that the local information at $\mathbf{x}$ that informed the choice of $\mathbf{p}_k$ (e.g., the TAYLOR series for f expanded at $\mathbf{x}_k$) remains valid. Large values of c_k are risky because they might step into a region where $f(\mathbf{x}_k + c_k\mathbf{p}_k)$ is not much smaller (or even larger) than $f(\mathbf{x}_k)$.

- We want c_k to be sufficiently large to make progress toward the optimal point. Small values of c_k are safe but can lead to glacial convergence.

Optimization algorithms typically impose two conditions to address these competing aims.

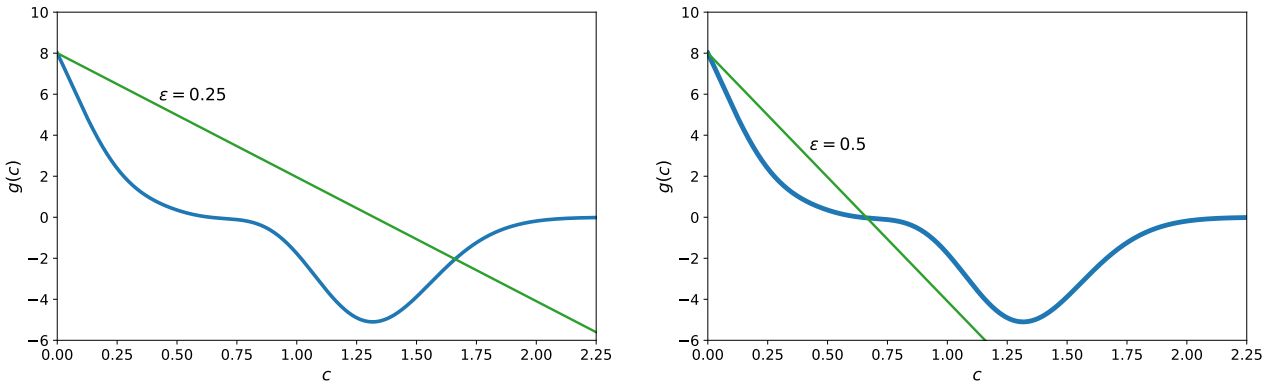


Figure 10.9: Illustration of the sufficient decrease condition, comparing two values of the parameter ε: $\varepsilon = 0.25$ (left) and $\varepsilon = 0.50$ (right). All values of c for which the blue curve $g(c) = f(\mathbf{x}_k + c\mathbf{p}_k)$ is under the green line $f(\mathbf{x}_k) + c(\varepsilon \mathbf{p}_k^T \nabla f(\mathbf{x}_k))$ satisfy the condition (10.4).

The sufficient decrease condition
Given a parameter ε $(0 < \varepsilon < 1)$ and descent direction $\mathbf{p}_k$, we say that a step-length c satisfies the *sufficient decrease* (or *Armijo*) condition provided

$$g(c) \leq f(\mathbf{x}_k) + c\Big(\varepsilon \mathbf{p}_k^T \nabla f(\mathbf{x}_k)\Big). \tag{10.4}$$

This condition is most easily appreciated through pictures, as illustrated in Figure **??**. The left-hand side of (10.4) is just the function $g(c)$ defined in (10.3). The right-hand side of (10.4) is a line that intersects $g(c)$ at $c = 0$ and has slope $\varepsilon \mathbf{p}_k^T \nabla f(\mathbf{x}_k)$ (which is negative, since $\mathbf{p}_k$ is a descent direction). At $c = 0$, $g'(c)$ will equal $-\mathbf{p}_k^T \nabla f(\mathbf{x}_k)$, and so the factor ε $(0 < \varepsilon < 1)$ modulates the slope, as seen in Figure **??**. The smaller ε, the larger the range of c values that will satisfy the sufficient decrease condition.

Notice that arbitrarily small $c > 0$ values satisfy the sufficient decrease condition, and thus we should add a second condition that ensures we do not take too small a choice of c. This condition will relate to the slope of the function $g(c)$, which we can compute as

$$g'(c) = \mathbf{p}_k^T \nabla f(\mathbf{x}_k + c\mathbf{p}_k).$$

The curvature condition
For a given step direction $\mathbf{p}_k$, we say the step length $c > 0$ satisfies the Wolfe conditions provided

$$\delta g'(0) \leq g'(c). \tag{10.5}$$

This condition requires that $g'(c)$ be at least factor of $1/\delta$ larger than the initial slope $g'(0)$. Since $g'(0) < 0$, this will imply, for one thing, that any c for which $g'(c) > 0$ satisfies the curvature condition.

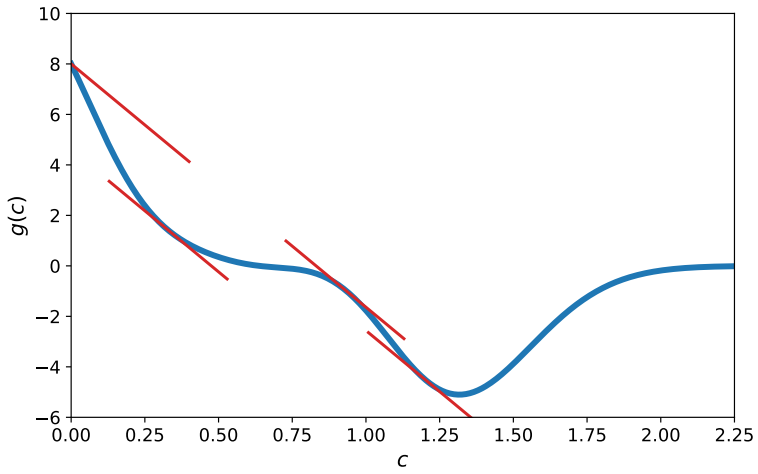


Figure 10.10: The curvature condition for $\delta = 0.4$. The red lines drawn along $g(c)$ have slope $\delta g'(0)$. Step-lengths $c > 0$ pass the curvature condition if $g'(c)$ is larger than $\delta g'(0)$.

Figure 10.10 shows $g(c)$ with lines of slope $\delta g'(0)$ drawn along $g(c)$ at a few values of c. If $g(c)$ is *less steep* than these red lines, then c satisfies the sufficient decrease condition. Study Figure 10.10, seeing if you can identify the ranges of c values that satisfy the curvature condition. Once you have worked out these values, turn forward to Figure 10.11, where you can check your answer against the values of c corresponding to the orange sections of the $g(c)$ plot.

10.3.3 The Wolfe conditions

Clearly, we seek values of c that satisfy both the sufficient decrease and curvature conditions, so we collect the two requirements together in the Wolfe conditions.

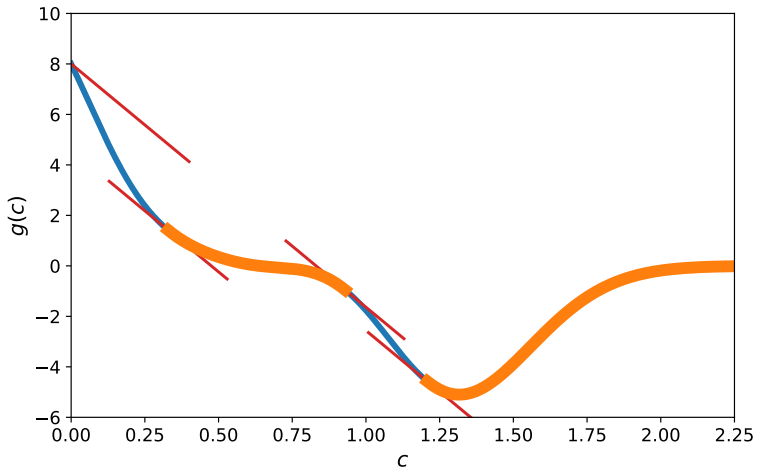


Figure 10.11: The sections of $g(c)$ colored in orange correspond to those c values that satisfy the curvature condition for $\delta = 0.4$.

The Wolfe conditions

Let $\mathbf{p}_k$ be a descent direction for $f : \mathbb{R}^n \to \mathbb{R}$ at the point $\mathbf{x}_k$, and define, for $c > 0$,

$$g(c) := f(\mathbf{x}_k + c\mathbf{p}_k)$$

having derivative

$$g'(c) = \mathbf{p}_k^T \left(\nabla f(\mathbf{x}_k + c\mathbf{p}_k)\right).$$

Let the parameters ε and δ satisfy

$$0 < \varepsilon < \delta < 1.$$

we say that a step-length c satisfies the Wolfe *conditions* provided c satisfies both the sufficient decrease condition

$$g(c) \leq f(\mathbf{x}_k) + c\left(\varepsilon \mathbf{p}_k^T \nabla f(\mathbf{x}_k)\right)$$

and the curvature condition

$$\delta g'(0) \leq g'(c).$$

Figure 10.12 shows, for our running example, the values of c that satisfy both the Wolfe conditions, highlighted by the gray regions on the plot. Notice that the curvature condition eliminates small values of c that satisfy the sufficient decrease condition, while the sufficient decrease condition eliminates the large values of c where the local information that informed the model near $c = 0$ are no longer describes the behavior of $g(c)$.

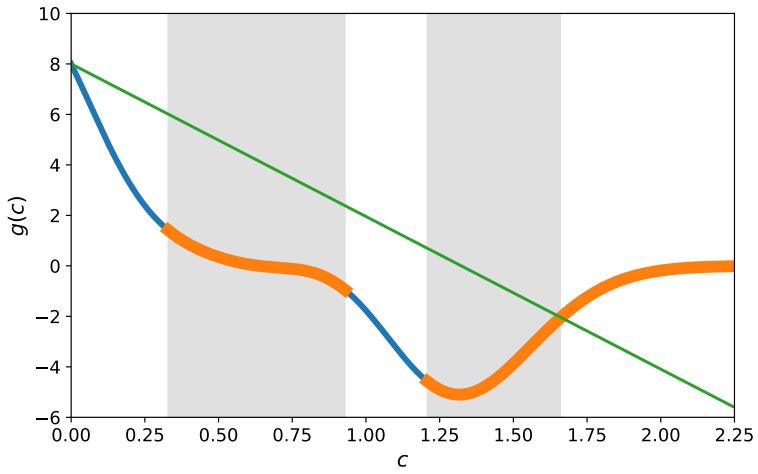


Figure 10.12: The Wolfe conditions: The c values that fall in the gray regions satisfy both the sufficient decrease condition ($\varepsilon = 0.25$) and the curvature condition ($\delta = 0.4$).

10.4 *Stochastic methods for large-scale problems*

10.5 *Newton's Method*

Given an approximation $\mathbf{x}_k \in \mathbb{R}^n$ to the minimum of $f : \mathbb{R}^n \to \mathbb{R}$, Newton's method seeks to construct an improved estimate $\mathbf{x}_{k+1}$ of the minimum by optimizing the quadratic Taylor approximation (10.1) to f at $\mathbf{x}_k$:

$$\phi(\mathbf{x}_k + \mathbf{p}) = \phi(\mathbf{x}_k) + \mathbf{p}^T \nabla f(\mathbf{x}_k) + \frac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x}_k)\mathbf{p}. \tag{10.6}$$

The goal is to find the value $\mathbf{p}_k$ of $\mathbf{p}$ that minimizes $\phi(\mathbf{x}_k + \mathbf{p})$, and use it to update the estimate to the minimum of f:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k.$$

To minimize ϕ, find its critical point: take the gradient of (10.6) *with respect to* $\mathbf{p}$ to obtain

$$\begin{aligned}\nabla_{\mathbf{p}} \phi(\mathbf{x}_k + \mathbf{p}) &= \nabla_p(\phi(\mathbf{x}_k)) + \nabla_p(\mathbf{p}^T \nabla f(\mathbf{x}_k)) + \nabla_p(\tfrac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x}_k)\mathbf{p}) \\ &= \mathbf{0} + \nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)\mathbf{p}.\end{aligned}$$

The critical point $\mathbf{p}_k$ of $\phi(\mathbf{x}_k + \mathbf{p})$ thus occurs when

$$\nabla^2 f(\mathbf{x}_k)\mathbf{p}_k = -\nabla f(\mathbf{x}_k).$$

Assuming the Hessian of f is invertible at $\mathbf{x}_k$, we can thus write

$$\mathbf{p}_k = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k).$$

Students typically first encounter Newton's method in a first calculus class, where it arises as a method for computing a zero of a single-variable function $f : \mathbb{R} \to \mathbb{R}$. One might recall the formula

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

This algorithm readily extends to minimization, which amounts to finding zeros of f', i.e., points where $f'(x) = 0$. For this task the iteration becomes

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

Notice this as the perfect one-

Starting from the initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, NEWTON's method seeks a local minimum of $f : \mathbb{R}^n \to \mathbb{R}$ through the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k), \tag{10.7}$$

assuming the Hessian $\nabla^2 f(\mathbf{x}_k) \in \mathbb{R}^{n \times n}$ is invertible at each step k.

The convergence of NEWTON's method if famously delicate: when $\mathbf{x}_0$ is sufficiently close to a local minimum, NEWTON's method typically converges *quadratically* to that minimum, meaning that it

doubles the number of correct digits at each iteration.

This is much faster than typical line search algorithms. However, the caveat that " $\mathbf{x}_0$ is sufficiently close to a local minimum," is crucial: otherwise NEWTON's method can behave erratically, converging to other critical points or diverging altogether.

10.6 Trust region methods