Mathematics of Scientific Computing

An Introduction with Matlab

Eric Hallman

PUBLISHED BY PUBLISHER

BOOK-WEBSITE.COM

First printing, March 2019

Contents

Part One

1. Computers Are Not Magic

1.1 Approximation Error

Let's begin with a simple physics experiment. Say we want to measure the gravitational acceleration near the Earth's surface (official value: $g = 9.80665$ m/s^2). To do so, we drop an object from a certain height h, measure the time t the object takes to hit the ground, and by using the formula

$$h = \frac{1}{2}gt^2, \tag{1.1}$$

find that $g = 2h/t^2$. For this particular experiment, we measure $\tilde{h} = 1.80$ m and $\tilde{t} = 0.59$ s, getting an approximation $\tilde{g} = 10.34$ m/s^2.

This is not the correct answer, so where did the error come from? Here are a few possible sources:

- Measurement error. Some of this could be due to the instruments: our ruler and watch can only make measurements to a certain precision, and perhaps they are not as accurate as we would hope. Some of this could be human error: we can't start and stop the watch precisely, and we may not have measured the height as accurately as we could have.
- Modeling error. For example, we ignored air resistance in our model and so Equation 1.1 is not quite correct. Furthermore, g is not constant over the surface of the Earth.
- Storing and manipulating numbers. The calculator we used only stores numbers to about 16 digits, so even if we knew h and t exactly we could not find the exact value of $2h/t^2$.
- Hardware faults. The calculator might be faulty either through age or through a design flaw, or perhaps a cosmic ray hit the calculator and changed the value of one of the numbers in its memory.

There are many problems for which we cannot expect to find an exact answer. What we would like to do instead is guarantee that the error falls below a certain threshold. What that threshold should be depends on the nature of the problem, but let's introduce two basic ways to measure error.

Definition 1.1.1 — Absolute Error. Let x be a scalar and let $\tilde{x}$ be an approximation to x. The **absolute error** is defined as $|x - \tilde{x}|$.

Definition 1.1.2 — Relative Error. Let x be a scalar and let $\tilde{x}$ be an approximation to x. The **relative error** is defined as $|x - \tilde{x}|/|x|$.

■ **Example 1.1** In our physics experiment, the true acceleration due to gravity was $g \approx 9.80665$ m/s^2, and our estimate was $\tilde{g} = 10.34$ m/s^2. The absolute error is

$$|g - \tilde{g}| = |9.80665 \text{ m/s}^2 - 10.34 \text{ m/s}^2| \approx 0.53 \text{ m/s}^2,$$

and the relative error is

$$\frac{|g - \tilde{g}|}{|g|} = \frac{|9.80665 \text{ m/s}^2 - 10.34 \text{ m/s}^2|}{|9.80665 \text{ m/s}^2|} \approx 0.054 = 5.4\%.$$

■

Which of these error measurements should we prefer? We will mostly use relative error in this book, but here are a few considerations:

- Relative error may not make much sense when using a scale without a *meaningful zero value*. If the temperature outside is 1°F and we estimate it to be 2°F, it would be strange to claim that our estimate is 100 percent off.
- It is possible to have a small absolute error and a large relative error, or vice versa. If the bond length of H_2O is $0.96 \cdot 10^{-10}$ m and our approximation is off by a single nanometer (10^{-9} m), then the relative error is enormous and probably unacceptable for most applications. On the other hand, an error of several miles is not so bad if we are estimating the distance from Earth to the Sun.
- Relative error is unitless and therefore insensitive to scaling (as long as zero remains fixed). If we changed the units of our physics experiment from meters/seconds to miles/hours or even furlongs/fortnights, the relative error would remain unchanged.
- Relative error can very nicely describe how numbers are stored and manipulated on most machines—we will explore this more when we cover floating point arithmetic.

Relative errors tell us about how many digits two numbers have in common.

Rule of Thumb 1.1.1 — Significant Digits. If $\tilde{x}$ is an approximation to a real number x and

$$\frac{|x - \tilde{x}|}{|x|} \leq 5 \cdot 10^{-k}$$

for some integer $k \geq 1$, then we may say that $\tilde{x}$ agrees with x to k digits.

■ **Example 1.2** If we approximate $x = 1.000$ by either $\tilde{x}_1 = 1.003$ or $\tilde{x}_2 = 0.997$, we will get a relative error of

$$\frac{|x - \tilde{x}|}{|x|} = 0.003 \leq 5 \cdot 10^{-3}.$$

It is therefore reasonable to say that both $\tilde{x}_1$ and $\tilde{x}_2$ approximate x to 3 digits, even though 0.997 and 1.000 do not technically have any digits in common. ■

■ **Example 1.3** If $x = 0$, then the relative error is zero when $\tilde{x} = 0$ and undefined otherwise. In other words, the only accurate approximation to 0 is 0. ■

Definition 1.1.3 If the relative error between x and $\tilde{x}$ is greater than 1, than our approximation is **garbage**[a]. We could have gotten a smaller relative error simply by guessing $\tilde{x} = 0$, and in fact $\tilde{x}$ might not even have the same sign as x!

[a] This is not a technical term.

■ **Example 1.4** If $x = 100$, $\tilde{x}_1 = 50$, and $\tilde{x}_2 = 200$, then

$$\frac{|x - \tilde{x}_1|}{|x|} = 0.5 \quad \text{and} \quad \frac{|x - \tilde{x}_2|}{|x|} = 1.$$

In some situations, you might consider $\tilde{x}_1$ and $\tilde{x}_2$ to be equally bad because they are both off by a factor of 2. If this is the case, the relative and absolute errors as we have defined them are not quite the right measures of accuracy. ■

1.1.1 Reporting Error

When reporting estimates and errors, one should follow the general principle of reporting quantities to an accuracy high enough to be informative, but not more than that. Exactly what this means depends on the situation, but we can provide a few simple rules of thumb. To start, most of the time we care only about the *order of magnitude* of the error rather than its exact value.

Rule of Thumb 1.1.2 Errors should generally be reported to at most one significant digit.

In the context of a numerical or physical experiment, error is sometimes reported in terms of an interval

$$(\text{estimate of } x) = \tilde{x} \pm \delta x,$$

where the (nonnegative) term δx represents the *uncertainty* in the estimate. This might mean that the interval $[\tilde{x} - \delta x, \tilde{x} + \delta x]$ is guaranteed to contain the true value x, but it is often used less precisely than that since guaranteed bounds frequently require δx to be too large to be useful. Often the bounds are used to represent what is called a 95% confidence interval, meaning that if we repeat the experiment many times then about 95% of such intervals will contain the true value x. In many practical situations, δx is just an estimate of the uncertainty, and so should not be reported to high precision.

Second, it is generally not useful to report additional digits in the estimate $\tilde{x}$ if these digits are more precise than the uncertainty in $\tilde{x}$.

Rule of Thumb 1.1.3 An estimate $\tilde{x}$ should generally be reported to roughly the same order of magnitude as the uncertainty.

■ **Example 1.5** A person estimates their weight to be 182.26 pounds, plus or minus 5.22 pounds. It would be more reasonable to report this estimate as 182 ± 5 pounds. ■

■ **Example 1.6** In competitive swimming, times are by FINA regulations measured to the hundredth of a second even though clocks are capable of measuring times to the thousandth of a second. One reason is that the length of a regulation pool can range between 50m and 50.03m, and even the length of a single pool can vary depending on the temperature of the water and surrounding environment. In a 50 meter race where swimmers average 2m/s, a 3cm difference in length can lead to a difference in time of 0.015 seconds. So even if times were measured to the thousandth of a second, the accuracy of the timer is not the limiting factor in assessing whether one swimmer is faster than another. ■

■ **Example 1.7** The standard acceleration due to gravity is equal to 9.80665 m/s^2 by definition, but it is meant to correspond roughly to the acceleration at 45 degrees latitude at sea level. It

approximates the actual acceleration anywhere on the Earth's survace to within half a percent relative error, or about 0.04 m/s^2. Without knowing anything about our specific location, it would be reasonable to use 9.80 ± 0.04 as an estimate. ■

1.2 Floating Point Representations

In everyday life, we use decimal (base 10) numbers. For example,

$$123.45 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}.$$

But we can represent numbers in other ways as well. In binary (base 2), for example, the only symbols we use are 0 and 1:

$$5.25 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = (101.01)_2.$$

This format is more natural for computers, which typically store information in terms of 0/1 bits at the hardware level.

■ **Example 1.8** How can we convert the number 25 from decimal to binary? One method is to find the largest power of 2 that is less than or equal to 25. This is 16 or 2^4, so we rewrite the number as $25 = 16 + 9$. Then $9 = 8 + 1$, so

$$25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (11001)_2.$$

■

■ **Example 1.9** Converting the number 101.101 from binary to decimal:

$$(101.101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 4 + 1 + 1/2 + 1/8 = 5.625.$$

■

1.2.1 Integer Formats

Armed with binary, we can store integers in the form of bits. Two common data types in Matlab are `uint8` and `int8`. The `uint8` format stores numbers as an **u**nsigned **8**-bit **int**eger. Since $2^8 = 256$, this means that it can store values ranging from 0 to 255.

If we want a format that can represent negative integers as well as positive, we can use the `int8` format instead. Matlab uses a representation system known as *twos complement*, but for simplicity (and for its relation to the format for storing floating point numbers) we will consider *offset binary* format, where the stored bit string f represents the integer $(f)_2 - 128$. Thus with offset binary, we can store integer values from -128 to 127 (`int8` has the same range).

■ **Example 1.10** In the `uint8` format, the bit string `01001101` represents the number $2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = 77$. For a signed integer using offset binary, the same string would represent the number $77 - 128 = -51$. ■

For many applications, 8 bits gives us all the precision we need—with grayscale images, for example, each pixel is typically encoded using 8 bits, with 0 representing black and 255 representing white. If we want to deal with larger integers, we can turn to `int16`, `int32`, or even `int64`[1].

	Unsigned		Signed	
# Bits	`intmin`	`intmax`	`intmin`	`intmax`
8	0	255	-128	127
16	0	65535	-32768	32767
32	0	4294967295	-2147483648	2147483647
64	0	$2^{64} - 1$	-2^{63}	$2^{63} - 1$

[1] These are all powers of 2 for the sake of making the hardware more efficient, but that's a subject for another course.

1.2.2 IEEE Floating Point Format

Integers aren't the only numbers we'd like to store, of course. More generally, we'd like a format for representing numbers that satisfies all of the following criteria:

- Numbers should be represented in terms of 0/1 bits.
- A single configuration of bits cannot represent more than one value.
- It should be simple to compare two numbers based on their bit representations. (The goal here is to minimize the cost of sorting data.)
- All numbers should take up the same number of bits, regardless of magnitude. (This allows the machine to do arithmetic faster, for hardware reasons.)
- We should be able to store real numbers to high precision. For example, Matlab by default stores pi as approximately 3.141592653589793.
- We should be able to store numbers with very large and very small magnitudes. Avogadro's number is about $6.02 \cdot 10^{23}$, the number of ways to shuffle a deck of cards is $52! \approx 8.0658 \cdot 10^{67}$, and Planck's constant is approximately $6.626 \cdot 10^{-34} J \cdot s$.

One format that satisfies all of these criteria is the IEEE 754 floating point format. The IEEE standard, first approved in 1985 and lasted updated in 2019, is the dominant standard for floating point arithmetic.

Definition 1.2.1 A **normalized** non-zero real floating point number in base 2 is represented as

$$(-1)^s (1.f)_2 \cdot 2^{e-c},$$

with three components.

- Sign $s = 0$ or $s = 1$
- Exponent e
- Fraction f.

The part $(1.f)_2$ is called the significand. The number c introduces a *bias* to the exponent, which allows us to store numbers with large or small magnitudes (here, e will be a nonnegative integer).

In IEEE format, we then store a floating point number as the following bit string, where the order is chosen to make comparisons fast.

s	e	f

The two most common binary formats are *single precision* and *double precision*. We will also define the non-standard *tiny precision* for the sake of examples and exercises.

Precison	Number of bits	s	e	f	c
Single	32	1	8	23	127
Double	64	1	11	52	1023
MA 402 Tiny	6	1	3	2	3

■ **Example 1.11** The number 25 can be written as $(-1)^0 (1.1001)_2 \cdot 2^4$, or equivalently,

$$25 = (-1)^0 (1.1001)_2 \cdot 2^{1027-1023},$$

so in double precision format it would be written as $0\ \underbrace{10000000011}_{2^{10}+2^1+2^0=1027}\ 1001\underbrace{0\ldots0}_{48}$. ■

R For normalized numbers, the leading 1 in the significand $(1.f)_2$ does not need to be stored explicitly. This effectively gets us an extra bit of storage for free.

■ **Example 1.12** The number 5.25 can be written as

$$5.25 = (-1)^0(1.0101)_2 \cdot 2^{1025-1023},$$

so in double precision format it would be written as `0` $\underbrace{\texttt{10000000001}}_{2^{10}+2^0=1025}$ $\texttt{0101}\underbrace{\texttt{0...0}}_{48}$. ■

■ **Example 1.13** The number -1 can be written as

$$-1 = (-1)^1(1.0)_2 \cdot 2^{1023-1023},$$

so in double precision format it would be written as `1` $\underbrace{\texttt{01111111111}}_{1023}$ $\underbrace{\texttt{0...0}}_{52}$. ■

Differences between Real and Floating Point Numbers

- There are uncountably many real numbers. By contrast there are finitely many floating point numbers: double precision uses 64 bits, and so cannot store more than 2^{64} numbers.
- There is no largest real number and there is no smallest positive real number. There is however, a largest floating point number and a smallest positive floating point number.
- Between any two real numbers there are uncountably many real numbers. Floating point numbers have gaps between them, and the gaps between numbers are not equally spaced!
- There are uncountably many irrational numbers among the reals. By contrast, *every* floating point number is rational and can be written in the form $m/2^n$ where m and n are integers.

Special Cases

A few numbers that don't fit into the normalized format. In all of these cases we will consider a number v represented by the triple (s,e,f) in double precision.

- Two zeros: if $e = 0$ and $f = 0$, then

  $$v = (-1)^s 0.$$

  This will not matter for most purposes as the two are treated as equal, but it can be useful for debugging. As an example, $1/0 = \infty$, but $1/(-0) = -\infty$.
- Denormalized numbers (underflow): if $e = 0$ and $f \neq 0$, then

  $$v = (-1)^s (0.f)_2 \cdot 2^{-1022}.$$

  These numbers are too small to be represented in normalized format. At least one important purpose they serve: without them, the claim "$a = b$ if and only if $a - b = 0$" would not hold for all pairs (a,b) in floating point arithmetic!
- Overflow: if $e = 2047 = 2^{11} - 1$ and $f = 0$, then

  $$v = (-1)^s \infty.$$

  Overflow results when a number is too large, such as 10^{1000}. In Matlab, use `Inf` or `inf`.
- Not a Number: if $e = 2047$ and $f \neq 0$, then

  $$v = \texttt{NaN}.$$

  Results from an invalid operation such as $0/0$ or $\infty - \infty$, and the value of f can be used for diagnostics or debugging. In Matlab, use `NaN` or `nan`.

■ **Example 1.14** The number $6\cdot 2^{-1026}$ is in the denormalized range, so we write it as $0.011\cdot 2^{-1022}$. Thus the representation in double precision is `0 00000000000 0110...0` (49 zeros). ■

■ **Example 1.15** There is no escaping from `NaN`: $0 * \texttt{NaN} = \texttt{NaN}$. ■

■ **Example 1.16** Consider the function

$$f(x) = 3 + \frac{1}{4+\frac{1}{1-x}}, \quad \text{or in Matlab,} \quad \texttt{f = @(x) 3+1./(4+1./(1-x))}.$$

IEEE arithmetic will return `f(1) = 3`, which is equal to $\lim_{x\to 1} f(x)$ even though $f(1)$ is itself undefined. ■

Summary

For double precision floating point numbers, the format is

$$v = \begin{cases} (-1)^s (1.f)_2 \cdot 2^{e-1023} & 1 \le e \le 2046 \\ (-1)^s (0.f)_2 \cdot 2^{-1022} & e = 0, f \ne 0 \\ (-1)^s \cdot 0 & e = 0, f = 0 \\ (-1)^s \infty & e = 2047, f = 0 \\ \texttt{NaN} & e = 2047, f \ne 0 \end{cases}.$$

For single precision floating point numbers, the format is

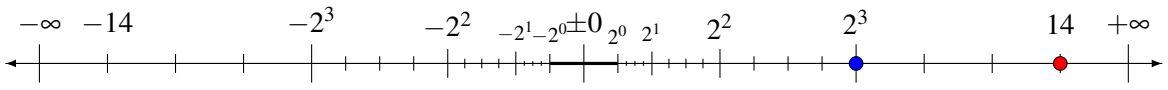
$$v = \begin{cases} (-1)^s (1.f)_2 \cdot 2^{e-127} & 1 \le e \le 254 \\ (-1)^s (0.f)_2 \cdot 2^{-126} & e = 0, f \ne 0 \\ (-1)^s \cdot 0 & e = 0, f = 0 \\ (-1)^s \infty & e = 255, f = 0 \\ \texttt{NaN} & e = 255, f \ne 0 \end{cases}.$$

For our nonstandard MA 402 Tiny precision, the format is

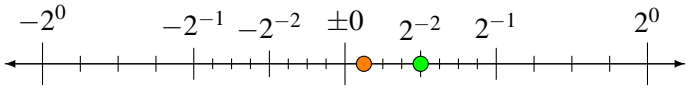
$$v = \begin{cases} (-1)^s (1.f)_2 \cdot 2^{e-3} & 1 \le e \le 6 \\ (-1)^s (0.f)_2 \cdot 2^{-2} & e = 0, f \ne 0 \\ (-1)^s \cdot 0 & e = 0, f = 0 \\ (-1)^s \infty & e = 7, f = 0 \\ \texttt{NaN} & e = 7, f \ne 0 \end{cases}.$$

1.2.3 Tiny Number Line

Here is the number line for our nonstandard MA 402 Tiny precsion, with the detail in the center omitted. Note that the numbers are not equally spaced, but the spacings are equal *between consecutive powers of 2*. For example, at the high end we have the consecutive numbers $4, 5, 6, 7, 8, 10, 12, 14, +\infty$. With the exception of the denormalized numbers, the spacing will increase by a factor of 2



Here is the center portion of the line. In this model, `realmin` (is 2^{-2} and all nonzero numbers with smaller magnitude are in the subnormal range ($e = 0, f \neq 0$).



A few points of interest:

- `realmax` (●) is the largest finite floating point number. In Tiny precision, `realmax` $= 14$. In double precision, `realmax` $= (2-2^{-52}) \cdot 2^{1023} \approx 10^{308}$.
  Its representation in double precision is `0 11111111110` $\underbrace{\texttt{11...11}}_{52}$.
- `realmin` (●) is the smallest *normalized* floating point number. In tiny precision, `realmin` $= 2^{-2}$. In double precision, `realmin` $= 2^{-1022} \approx 10^{-308}$.
  Its representation in double precision is `0 00000000001` $\underbrace{\texttt{00...00}}_{52}$.
- `flintmax` (●) is the largest consecutive integer. In tiny precision, `flintmax` $= 8$. In double precision, `flintmax` $= 2^{53} \approx 9 \cdot 10^{15}$. Integer arithmetic may not be error-free when working with numbers larger than `flintmax`.
- In general, `eps`(x) is the distance from x to the next floating point number that is larger in magnitude. The quantity

  $$\texttt{eps} = \varepsilon = \texttt{eps}(1)$$

  is often referred to as **machine precision**, and represents the distance between 1 and the next larger floating point number. In tiny precision, `eps` $= 0.25$. In double precision, `eps` $= 2^{-52} \approx 2.22 \cdot 10^{-16}$.
  The double-precision representation of $1+\varepsilon$ is `0 01111111111` $\underbrace{\texttt{00...00}}_{51}$1.
- `eps`(0) (●) is the smallest positive floating point number. In tiny precision, `eps`$(0) = 2^{-4}$. In double precision, `eps`(0) = `eps` $\cdot$ `realmin` $= 2^{-52} \cdot 2^{-1022} = 2^{-1074} \approx 10^{-324}$.
  The double-precision representation of `eps`(0) is `0 00000000000` $\underbrace{\texttt{00...00}}_{51}$1.

A table of all nonnegative numbers in MA 402 Tiny format is given in Figure 1.1. Note that as we go from one number to the next larger one, the binary representation increments like an odometer. Thus just by looking at the bit representation, we can easily tell which of two numbers is larger.

Note also that although the numbers on the line are not equally spaced in general, the numbers *between consecutive powers of two* are equally spaced, and the spacing doubles with each power of two (except for the subnormal range). More precisely, between normalized numbers 2^k and 2^{k+1} the spacing is equal to $2^k\varepsilon = \texttt{eps}(2^k)$, and there are exactly ε^{-1} numbers in the range $[2^k, 2^{k+1})$.

1.2.4 Notable Problems in Representing Numbers

- In 1996, the Ariane 5 rocket veered off its flight path soon after its launch and exploded. It was found that the failure occurred because the software attempted to convert a 64-bit floating point number (information about the horizontal velocity of the rocket) to a 16-bit signed integer and failed due to overflow. The rocket and its cargo were valued at approximately $500 million.
- In 1994, a hardware bug was discovered for Pentium processors that would occasionally lead to incorrect results when dividing two numbers. Intel ultimately recalled the processors, a move that cost the company hundreds of millions of dollars.

Number	Representation	Notes
0	0 000 00	Zero
1/16	0 000 01	eps(0)
1/8	0 000 10	
3/16	0 000 11	
1/4	0 001 00	realmin
5/16	0 001 01	
3/8	0 001 10	
7/16	0 001 11	
1/2	0 010 00	
5/8	0 010 01	
3/4	0 010 10	
7/8	0 010 11	
1	0 011 00	1
1.25	0 011 01	$1+\varepsilon$
1.5	0 011 10	
1.75	0 011 11	
2	0 100 00	
2.5	0 100 01	
3	0 100 10	
3.5	0 100 11	
4	0 101 00	
5	0 101 01	
6	0 101 10	
7	0 101 11	
8	0 110 00	flintmax
10	0 110 01	
12	0 110 10	
14	0 110 11	realmax

Figure 1.1: Table of all nonnegative MA 402 Tiny numbers

- The Y2K bug arose because many programs represented the year using only two digits. As a result, the year 2000 would be indistinguishable from 1900, which could lead to bugs when trying to order events chronologically. Worldwide, several years and hundreds of billions of dollars were spent fixing this potential problem.
- The Year 2038 problem: many systems represent time as a signed 32-bit integer counting the number of seconds passed since 00:00:00 UTC on January 1, 1970. This format will be not be able to encode times after 03:14:07 UTC on January 19, 2038.
- The *Civilization II* Gandhi bug (not quite as serious as the Ariane 5 failure.)

  - Each AI in the computer game has an "aggression rating", stored as an unsigned 8-bit integer (i.e., with a value between 0 and 255) on a scale of 1 to 10.
  - Gandhi, the most peaceful by nature, starts with an aggression rating of 1.
  - When a civilization adopts democracy, it changes the aggression rating by -2.
  - Different programming languages handle integer overflow in various ways. In this case, the number goes below zero and "wraps around", leading to the result $1-2=255$.
  - Result: shortly after adopting democracy, Gandhi nukes everyone and everything.

1.3 Floating Point Arithmetic

Most numbers cannot be represented exactly in terms of floating point numbers (there are only finitely many!), so when we convert a real number such as π into floating point format there will necessarily be some amount of *roundoff error*.

Definition 1.3.1 For the IEEE floating point format, the quantity

$$\mathtt{u} = \mathtt{eps}/2$$

is called **unit roundoff**. In double precision, $u = 2^{-53} \approx 1.11 \cdot 10^{-16}$.

Theorem 1.3.1 For a real numbers x, let $\mathrm{fl}(x)$ be the floating point representation of x. If $\mathtt{realmin} \le |x| \le \mathtt{realmax}$, then

$$\mathrm{fl}(x) = x(1+\delta) \quad \text{where} \quad |\delta| \le \mathtt{u}.$$

In essence, for any x in the normal range (approximately $[10^{-308}, 10^{308}]$), the floating point representation of x will differ from x by a small multiplicative factor.

The bound in Theorem 1.3.1 is equivalent to the bound

$$|\mathrm{fl}(x) - x| \le |x|\mathtt{u},$$

which implies that when x is large the *absolute* roundoff error could also be large. The *relative* error, however, will always be small, since when $x \neq 0$ this bound is equivalent to the bound

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \le \mathtt{u}.$$

Thus when working with floating point numbers, it is often natural to use the relative error. Since $\mathtt{u} \approx 1.11 \cdot 10^{-16}$ in double precision, we can say that $\mathrm{fl}(x)$ agrees with x to about 16 digits.

■ **Example 1.17** In IEEE double precision, π is rounded as

$$\mathrm{fl}(\pi) = \underline{3.141592653589793}115\ldots,$$

where the correct digits are underlined. The absolute error is

$$|\mathrm{fl}(\pi) - \pi| \approx 1.22 \cdot 10^{-16},$$

and the relative error is

$$\frac{|\mathrm{fl}(\pi) - \pi|}{|\pi|} \approx 3.90 \cdot 10^{-17}.$$

Note that the relative error is smaller than $\mathtt{u}$. ■

When x is not in the normal range, the bound of Theorem 1.3.1 is no longer guaranteed to hold.

- (**Overflow**) If $|x| \ge \mathtt{realmax} + \mathtt{eps}(\mathtt{realmax})/2 = (1 - 2^{-54})2^{1024}$, then $\mathrm{fl}(x) = \pm\mathtt{Inf}$.
- (**Gradual underflow**) If $\mathtt{eps}(0)/2 < |x| < \mathtt{realmin} - \mathtt{eps}(0)/2$, then $\mathrm{fl}(x)$ will be a denormalized floating point number, possibly with a large relative error.
- (**Underflow to zero**) If $|x| \le \mathtt{eps}(0)/2$, then $\mathrm{fl}(x) = 0$.

■ **Example 1.18** Let $x = \mathtt{eps}(0)$, the smallest positive floating point number. With IEEE rounding, $\mathrm{fl}(3x/2) = 2x$, which has a relative error of 1/3. ■

1.3.1 Rounding Rules

Even if numbers x and y can be represented exactly in floating point format, there is no guarantee that we will be able to exactly represent numbers such as $x+y$, $1/x$, xy, x/y, or $\sqrt{x}$.

- The number 3 can be represented exactly, but 1/3 cannot.
- The numbers 1 and 10^{20} can both be represented exactly, but $10^{20}+1$ cannot.
- The number $(1+\varepsilon)$ can be represented exactly, but $(1+\varepsilon)^2 = 1+2\varepsilon+\varepsilon^2$ cannot.

The IEEE system for rounding these numbers is as follows.

Fact 1.3.2 For any real number x satisfying $|x| \leq \texttt{realmax}$, $\text{fl}(x)$ is equal to the floating point number closest to x.

In the event of a tie, x is rounded to the number whose IEEE representation has a final bit of zero.

The tiebreaking rule is known as *round to even*, where "even" in this context means that the last bit is zero. Numbers on the IEEE number line always alternate strictly between "odd" and "even" in this sense, so this rule always gives a unique answer.

R A number can be "even" in the sense that its final bit is zero without being an integer divisible by 2, and vice versa. In the MA 402 Tiny system, for instance, the number 14 is represented as `0 110 11` and 3 is represented as `0 100 10` (see Figure 1.1).

When dealing with the elementary arithmetic operations, a similar rule holds.

Fact 1.3.3 Let x and y be normalized floating point numbers. For any elementary arithmetic operation $\circ \in \{+,-,\times,/\}$, the floating point operation $\text{fl}(x \circ y)$ is equal to the *exact* value of $x \circ y$, rounded correctly. Furthermore, the floating point operation $\text{fl}(\sqrt{x})$ is equal to the *exact* value of $\sqrt{x}$, rounded correctly.

It is worth emphasizing that the statement above does not necessarily hold for other functions, but the IEEE standard requires it of the elementary arithmetic operations and the square root operator.

Corollary 1.3.4 Let x and y be normalized floating point numbers. For any elementary arithmetic operation $\circ \in \{+,-,\times,/\}$, if $\texttt{realmin} \leq |x \circ y| \leq \texttt{realmax}$, then

$$\text{fl}(x \circ y) = (x \circ y)(1+\delta) \quad \text{where} \quad |\delta| \leq \mathtt{u}.$$

■ **Example 1.19** In exact arithmetic, $\sqrt{1+\varepsilon}$ is slightly less than $1+\varepsilon/2$. Since this number is closer to 1 than the next number $1+\varepsilon$, it gets rounded down to 1. ■

■ **Example 1.20** The number $1+\varepsilon/2$ is exactly halfway between 1 and $1+\varepsilon$, so the round-to-even rule applies. Since the IEEE representation of 1 has zero as its final bit (similar to Example 1.13), $1+\varepsilon/2$ gets rounded down to 1. ■

■ **Example 1.21** The numbers 3 and $1+\varepsilon$ can both be stored exactly, but $3(1+\varepsilon)$ cannot. It falls halfway between the "odd" number $3+2\varepsilon$ and the "even" number $3+4\varepsilon$, and so gets rounded up to $3+4\varepsilon$. ■

Fact 1.3.5 Let x be a normalized floating point number, and let p be an integer satisfying

$$\texttt{realmin} \leq \{|x|, 2^p, |2^p \cdot x|\} \leq \texttt{realmax}.$$

Then multiplication by a power of 2 is computed without any rounding error:

$$\mathrm{fl}(x \cdot 2^p) = x \cdot 2^p,$$

because only the exponent of x changes and the fractional bits of x remain the same.

■ **Example 1.22** In MA 402 Tiny arithmetic, $1.75 \cdot 8 = 14$ with no rounding error. The representation of 1.75 is `0 011 11` and the representation of 14 is `0 110 11`, which differs from 1.75 only in the exponent. ■

It is also worth emphasizing that Fact 1.3.3 applies specifically to the *binary* operations $\{+, -, \times, /\}$. For more complicated operations such as computing the sum of a list of numbers, the result will typically be rounded after each binary operation. As a result, the order of operations can make a difference on a machine even when it does not matter in exact arithmetic.

■ **Example 1.23** Consider the problem of finding the sum 1 + 7 + 2 in MA 402 Tiny arithmetic (see Figure 1.1). If the sum is computed from left to right, then we find that $\mathrm{fl}(1+7) = \mathrm{fl}(8) = 8$ and $\mathrm{fl}(8+2) = \mathrm{fl}(10) = 10$, the exact answer.

If the sum is computed from right to left, we instead get that $\mathrm{fl}(7+2) = \mathrm{fl}(9) = 8$ by the round-to-even rule, and from there that $\mathrm{fl}(1+8) = \mathrm{fl}(9) = 8$ by the same rule. Thus in Tiny precision, the computations $(1+7)+2$ and $1+(7+2)$ yielded different results! ■

■ **Example 1.24** In IEEE double precision, $(1+\varepsilon/2)+\varepsilon/2 = 1$, but $1+(\varepsilon/2+\varepsilon/2) = 1+\varepsilon$. ■

■ **Example 1.25** In IEEE double precision $\mathrm{fl}((\pi \cdot 13)/13) \neq \mathrm{fl}(\pi)$. ■

R Some hardware permits a *fused multiply-add* operation where expressions of the form $xy+z$ are computed with a single rounding operation. An unfused multiply-add, by contrast, would round the product xy, then add z and round again.

1.4 Big Oh Notation

Before moving on to the next chapter, we should introduce a helpful notational tool for asymptotic analysis of algorithms. When discussing how to report estimates and errors in Section 1.1.1 we mentioned the principle that only the informative digits should be reported. The notation here will allow us to do the same for algebraic expressions. A much fuller treatment can be found in [8], but we provide at least the details necessary for this course.

Definition 1.4.1 — Asymptotic Hierarchy. Positive functions f and g on the natural numbers are said to have different *asymptotic growth* if one tends to infinity faster than the other. Formally, we say that

$$f(n) \prec g(n)$$

if $\lim_{n\to\infty} f(n)/g(n) = 0$.

■ **Example 1.26** Looking at powers of n, we find that

$$\cdots \prec \frac{1}{n^2} \prec \frac{1}{n} \prec 1 \prec n \prec n^2 \prec \cdots$$

More generally, $n^\alpha \prec n^\beta$ if and only if $\alpha < \beta$. Note that these functions do not necessarily have to "tend to infinity" as $n \to \infty$. ■

Theorem 1.4.1 — Hierarchy of Growth. For any positive constants α and β,

$$1 \prec \ln(n) \prec n^{\alpha} \prec e^{\beta n} \prec n!,$$

where $n!$ is the factorial of n (Definition 4.2.8).

Fact 1.4.2 Multiplication by a nonzero constant does not affect the ordering of functions. If $f(n) \prec g(n)$ and $c > 0$, then $cf(n) \prec g(n)$.

The notation does not specify just how quickly the limit of the ratio will tend to zero, and it is possible that it will not do so soon enough for the precedence relation to be of much use to us.

■ **Example 1.27** While it is true that

$$n^{100} \prec (1.01)^n,$$

the first term will be larger when $2 \leq n \leq 117308$. By the time the second term overtakes the first, both quantities are larger than 10^{500}, which is larger than `realmax` in double precision. ■

Another useful bit of notation requires only that $g(n)$ grow *at least as fast* as $f(n)$.

Definition 1.4.2 — Big Oh. For functions f and g on the natural numbers, we may say that

$$f(n) = \mathcal{O}(g(n)) \qquad (\text{as } n \to \infty)$$

if there exist constants C and N such that

$$|f(n)| \leq C|g(n)| \qquad \text{whenever} \quad n \geq N. \tag{1.2}$$

The relation is commonly spoken as "f is big-Oh of g".

The key idea behind this notation is that it allows us to describe an expression by its fastest-growing term. Doing so can drastically simplify complicated expressions and give us a quick idea of roughly how fast a function increases.

■ **Example 1.28** Let $f(n)$ denote the sum of the first n natural numbers,

$$f(n) = \sum_{i=1}^{n} i = \frac{n^2+n}{2}.$$

Then

$$\frac{n^2+n}{2} = \frac{1}{2}n^2 + \frac{1}{2}n \leq \frac{1}{2}n^2 + \frac{1}{2}n^2 = n^2,$$

so we may say that $f(n) = \mathcal{O}(n^2)$ since 1.2 holds with $C = 1$ and $N = 1$. ■

Fact 1.4.3 If $f(n) \prec g(n)$, then $f(n) = \mathcal{O}(g(n))$. The converse is not necessarily true.

■ **Example 1.29** Letting $f(n)$ be the sum of the first n natural numbers as in Example 1.28, it is correct to say that $f(n) = \mathcal{O}(n^{10})$, but this bound is unnecessarily loose. ■

Fact 1.4.4 If $f(n) = \mathcal{O}(1)$, then f is bounded: there is a constant C such that $|f(n)| \leq C$ for all n. Conversely, if f is bounded then $f(n) = \mathcal{O}(1)$.

■ **Example 1.30** The sine function is bounded, so $\sin(n) = \mathcal{O}(1)$. ■

The big-Oh term can also be used as part of a larger expression.

■ **Example 1.31** Letting $f(n)$ be the sum of the first n natural numbers as in Example 1.28, we may say more precisely that

$$f(n) = \frac{1}{2}n^2 + \mathcal{O}(n),$$

because $|f(n) - n^2/2| \leq Cn$ for $C = 1/2$ and $n \geq 1$. In this way the big-Oh term can suppress the "unimportant" information, leaving us with an increasingly accurate approximation to $f(n)$. When $n = 100$, for example, the approximation $f(n) \approx n^2/2$ has a relative error of less than 0.01. ■

We are also interested in the rate at which functions tend to zero, and introduce a second context in which big-Oh notation can be used.

Definition 1.4.3 — Big Oh. For functions f and g on the real numbers, we may say that

$$f(x) = \mathcal{O}(g(x)) \qquad (\text{as } x \to 0)$$

if there exist constants C and ε such that

$$|f(x)| \leq C|g(x)| \qquad \text{whenever} \quad |x| \leq \varepsilon. \tag{1.3}$$

■ **Example 1.32** As x tends to zero,

$$\left|\frac{x^2 + x}{2}\right| \leq \frac{1}{2}x^2 + \frac{1}{2}|x| \leq |x|,$$

where the first inequality uses the triangle inequality (Theorem 2.1.4) and the second holds when $|x| \leq 1$. Thus the function is $\mathcal{O}(x)$, since Equation 1.3 holds with $C = 1$ and $\varepsilon = 1$.

Note that this different from the behavior of $f(n) = (n^2 + n)/2$. As n tends to infinity n^2 will grow more quickly than n, but as x tends to zero x^2 will *shrink* much more quickly than x. ■

1.4.1 Abuses of Notation

It should be noted that the equality sign in an expression such as

$$3n^2 + 7n + 4 = \mathcal{O}(n^2)$$

does not really mean that the two sides are equal, and the expression should never be written with the sides reversed. If we took the notion of equality at face value, we could take statements such as $n = \mathcal{O}(n^2)$ and $n^2 = \mathcal{O}(n^2)$ and reach false conclusions such as $n = n^2$. We *can* write statements with big-Oh notation on both sides, such as

$$3n^3 + \mathcal{O}(n^2) = \mathcal{O}(n^3),$$

which can be translated roughly as

$$3n^3 + f(n) = \mathcal{O}(n^3), \qquad \text{for all functions } f(n) = \mathcal{O}(n^2).$$

The equality sign still gets used largely due to tradition, but it does help at an intutive level when manipulating expressions.

A second common abuse is to say that $f(n) = \mathcal{O}(g(n))$ and in doing so imply that the growth of $f(n)$ is also bounded *below* by that of $g(n)$. For example, one might say

"Solving an $n \times n$ linear system by Gaussian elimination takes $\mathcal{O}(n^3)$ operations"

but taken literally, the statement would be correct even if the algorithm required only n^2 or even n operations. Computer scientists will address lower bounds by defining a few more terms.

Definition 1.4.4 — Big Omega. For functions f and g on the natural numbers, we may say that

$$f(n) = \Omega(g(n)) \qquad (\text{as } n \to \infty)$$

if there exist constants $C > 0$ and N such that

$$|f(n)| \geq C|g(n)| \qquad \text{whenever} \quad n \geq N.$$

Fact 1.4.5 $f(n) = \Omega(g(n))$ if and only if $g(n) = \mathcal{O}(f(n))$.

■ **Example 1.33** $(n^2+n)/2$ is $\mathcal{O}(n^3)$ but not $\Omega(n^3)$. ■

Definition 1.4.5 — Big Theta. For functions f and g on the natural numbers, we may say that

$$f(n) = \Theta(g(n)) \qquad (\text{as } n \to \infty)$$

if $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

■ **Example 1.34** The function $n^3 + 3n^2 + 4n + 7$ is $\Theta(n^3)$. ■

Being the lazy mathematicians that we are, we will continue to say $f(n) = \mathcal{O}(g(n))$ when we really mean that $f(n) = \Theta(g(n))$, at least in the context of analyzing the cost of algorithms.

While we're on the subject of misused terminology, here's one that gets abused all too frequently:

Definition 1.4.6 — Exponential Growth and Decay. *Exponential growth* refers to functions of the form $f(n) = \alpha e^{\beta n}$ where $\beta > 0$. If $\beta < 0$ instead, then the behavior is described as *exponential decay*.

Do not use these terms to refer to the behavior of functions such as n^2 or $1/n$.

1.4.2 Rules of Manipulation

It can be rather tedious to show directly that a function satisfies constraints of the form 1.2 or 1.3, so we introduce a few rules for manipulating big-Oh expressions.

Theorem 1.4.6 The following relations all hold:

$$\begin{aligned}
f(n) &= \mathcal{O}(f(n)), \\
c \cdot f(n) &= \mathcal{O}(f(n)), \quad \text{for any constant } c, \\
\mathcal{O}(f(n)) + \mathcal{O}(g(n)) &= \mathcal{O}(|f(n)| + |g(n)|), \\
n^\alpha &= \mathcal{O}(n^\beta), \quad \text{whenever } \alpha \leq \beta, \\
\mathcal{O}(\mathcal{O}(f(n))) &= \mathcal{O}(f(n)), \\
\mathcal{O}(f(n))\mathcal{O}(g(n)) &= \mathcal{O}(f(n)g(n)), \\
\mathcal{O}(f(n)g(n)) &= f(n)\mathcal{O}(g(n)),
\end{aligned}$$

in the sense that the left-hand expression may be replaced by the right-hand expression and still retain the truth of a statement.

■ **Example 1.35** Carrying on with the same function as in Example 1.28, we can now say that

$$(n^2+n)/2 = \frac{1}{2}n^2 + \frac{1}{2}n = \mathcal{O}(n^2) + \mathcal{O}(n^2) = \mathcal{O}(n^2),$$

without having to hunt for any constants C and N to make the bound 1.2 hold. ■

1.4.3 Applications

Big-Oh notation is extremely useful when dealing with power series. If we have a function f with absolutely convergent Taylor series at x,

$$f(x+h) = \sum_{k=0}^{\infty} \frac{h^k}{k!} f^{(k)}(x),$$

then we can truncate the function to a desired degree n to get the approximating polynomial

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \cdots + \frac{h^n}{n!} f^{(n)} x + \mathcal{O}(h^{n+1}).$$

The error term $\mathcal{O}(h^{n+1})$ tells us that there exist constants C and ε (which depend on n) such that the approximation error is bounded by Ch^{n+1} when $|h| \leq \varepsilon$. It does not tell us what the constant C is or how small ε needs to be.

■ **Example 1.36** For $|x| < 1$, the power series

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots$$

converges absolutely. As $x \to 0$, we can say

$$\frac{1}{1-x} = \mathcal{O}(1),$$
$$\frac{1}{1-x} = 1 + \mathcal{O}(x),$$
$$\frac{1}{1-x} = 1 + x + \mathcal{O}(x^2),$$

and so on to any degree we like. ■

From the rules for manipulating big-Oh notation mentioned in the previous section, we find a very handy shortcut for manipulating truncated power series.

Theorem 1.4.7 — Polynomial Truncation. Let p_n in general denote the n-th degree truncation of a power series p. Then for power series p and q,

$$(p+q)_n = (p_n + q_n)_n,$$
$$(pq)_n = (p_n q_n)_n.$$

That is to say, to find the truncation of a sum or product, we can truncate the individual series first, then find the sum or product, then truncate again. We can save a lot of work this way!

■ **Example 1.37** What is the degree-2 truncation of the Taylor series for $1/(1-x)^2$? We can say that

$$\begin{aligned}\left(\frac{1}{1-x}\right)^2 &= \left(1 + x + x^2 + \mathcal{O}(x^3)\right)^2 \\ &= 1 + 2x + 3x^2 + 2x^3 + x^4 + 2(1 + x + x^2)\mathcal{O}(x^3) + \mathcal{O}(x^3)^2 \\ &= 1 + 2x + 3x^2 + 2x^3 + x^4 + \mathcal{O}(x^3) + \mathcal{O}(x^3) \\ &= 1 + 2x + 3x^2 + \mathcal{O}(x^3).\end{aligned}$$

■

Truncating power series is very useful when working to simplify bounds involving roundoff errors, and we'll see more of this in the next chapter.

■ **Example 1.38** If $|\delta| \leq \mathtt{u}$, then

$$\frac{1}{1+\delta} = 1-\delta+\delta^2-\cdots \leq 1+\mathtt{u}+\mathtt{u}^2+\cdots = 1+\mathtt{u}+\mathscr{O}(\mathtt{u}^2).$$

■

Formally, in this example the big-Oh would be with respect to $\mathtt{u}$ as $\mathtt{u} \to 0$. This interpretation may seem somewhat odd, since typically we work exclusively in one precision (usually single or double). In practice, a simpler interpretation is that $\mathtt{u}$ is small enough for any and all terms contained in $\mathscr{O}(\mathtt{u}^2)$ to be safely ignored. In double precision, this is a pretty safe assumption. For low-precision types such as half precision ($1+5+10=16$ bits) or `bfloat16` ($1+8+7=16$ bits), more careful error analysis might be necessary.

In the case of algorithms, we also care about the *time* required to run an algorithm as well as the amount of data we will need to store in *memory*.

Definition 1.4.7 A *flop* is a **fl**oating **po**int operation, such as $\{+,-,\times,/\}$.

Definition 1.4.8 In the context of scientific computing, computer performance is often measured in *FLOPS* (or *flop/s*), which stands for *flops per second*.

It is in general difficult to predict how long a computer will take to execute any particular algorithm, and the number of flops required provides only a very crude estimate. First of all, a computer will not necessarily reach peak performance for all of the operations it carries out. Second of all, it takes time to move data between main memory and the processor, and computers have multiple levels of memory. Third, most modern computers have multiple cores that can carry out instructions in parallel, but not all algorithms can be parallelized equally well. Nonetheless, it does give us at least a rough idea of what problem sizes the computer might be capable of handling.

For a sense of context, a modern desktop computer might have 32GB (gigabytes, or 10^9 bytes) of memory and a peak speed of 500 GFLOPS. At the top end, the Fugaku supercomputer at the Riken Center for Computational Science in Japan attained a performance of 416 petaFLOPS (10^{15} FLOPS) on the LINPACK benchmark. So given that a double precision number takes 8 bytes (a byte is 8 bits, so $8\times 8 = 64$ bits), you could store a vector with up to a billion entries or so on your desktop computer. An $n\times n$ matrix has n^2 entries, so the largest dense square matrix you can store will have dimensions around $n \approx 30,000$. Want to store an $n\times n\times n$ cube of numbers? You're down to $n = 1000$ now.

Similarly for arithmetic: generously assuming that your computer carries out 500 billion operations per second, finding the sum of $n = 10,000$ numbers (an $\mathscr{O}(n)$ algorithm) would be dirt cheap. An algorithm requiring n^2 operations would require about 0.0002 seconds, and an algorithm requiring n^3 operations would take about 2 seconds. For a given $\mathscr{O}(n^3)$ algorithm, increasing the problem size by a factor of 10 will increase the number of flops by a factor of 1000. Try $n = 100,000$ (even if you had the memory to do it) and your n^3 operations now require half an hour; bump n up to a million and you'll need three weeks!

1.5 Matlab Commands

Getting Started

- `help NAME`: Display information on how to use `NAME`.
- `doc NAME`: Open the documentation page for `NAME`. Provides more detailed information than `help`.

- `clc`: Clear the command window.
- `clear`: Clear variables and functions from memory.

Basic operations

- `x^y`: Returns `x` to the power `y`.
- `1.23eK`: Returns $1.23 \cdot 10^K$, where K is an integer.
- `abs(x)`: Returns the absolute value of `x`. If the input is a vector or matrix, the absolute value is taken for each individual element.
- `sign(x)`: Returns 1/0/(-1) if `x` is positive/zero/negative.
- `log(x)`: Natural logarithm of `x`. See `log2` and `log10` for the base 2 and base 10 logarithms.
- `round(x)`: Rounds `x` to the nearest integer. Has other options that allow you to round `x` to a given number of decimal places or significant digits.
- `floor(x)`, `ceil(x)`, `fix(x)`: Round `x` down, up, or toward zero.
- `mod(x,y)`: Returns the remainder when `x` is divided by `y`, and has the same sign as `y`.
- `rem(x,y)`: Similar to `mod(x,y)`, but keeps the same sign as `x` and handles the case `y=0` differently.

Floating point representations

- `format long`: Changes the display to show numbers to 15 decimal places. Does not affect how the underlying computations are done.
- `format short`: Default setting for Matlab, fixed point with 5 digits.
- `eps(x)`: Returns the distance between `x` and the next larger (in magnitude) floating point number.
- `eps`: Equal to `eps(1)`, or 2^{-52}. In double precision, $1+\varepsilon$ is the smallest number greater than 1.
- $\pm$`Inf`: Returns $\pm\infty$.
- `nan`: Stands for Not-a-Number. Useful for exception handling.
- `realmin`: The smallest normalized floating point number in double precision, equal to 2^{-1022}. The smallest nonzero number is `eps*realmin`.
- `realmax`: The largest finite floating point number in double precision, equal to $(2-2^{-52}) \cdot 2^{1023}$.
- `flintmax`: The largest consecutive integer in double precision, equal to 2^{53}. Above this level not all integers can be represented exactly.
- `isnan`, `isinf`, `isfinite`: Return whether a given number is NaN, infinite, or finite, respectively.

Numeric formats

- `dec2bin(x)`: Converts nonnegative integer `x` to a binary string.
- `bin2dec(s)`: Converts binary string `s` to decimal value.
- `bitget(x,BIT)`: returns the value of the bit at position `BIT` in `x`.
- `single(x)`: Converts `x` to single precision.
- `double(x)`: Converts `x` to double precision (the default).
- `eps('single')`: Equal to `eps(single(1))`, or 2^{-23}.
- `realmin('single')`: The smallest normalized floating point number in single precision, equal to 2^{-126}.
- `realmax('single')`: The largest finite floating point number in single precision, equal to $(2-2^{-23}) \cdot 2^{127}$.

1.6 Exercises

1. It is known that $\pi \approx 22/7$ is a simple rational approximation of the value of pi. What are the absolute and relative errors for this approximation?
2. (Taylor [21]) (a) A digital voltmeter reads voltages to the nearest thousandth of a volt. What will be its percent uncertainty in measuring a voltage of approximately 3 volts? (b) A digital balance reads masses to the nearest hundredth of a gram. What will be its percent uncertainty in measuring a mass of approximately 6 grams?
3. In the 1965 song *New Math*, Tom Lehrer solves the problem $(342)_8 - (173)_8$. What is the solution of this problem, written in base 8? What is the translation of this problem to base 10?
4. A hilarious joke:

   > There are only 10 types of people in the world: those who understand binary and those who don't.

   Please explain the joke.
5. Count to 30 using each of the following methods. Which method do you find most convenient?
   (a) On both hands, using a base 6 system.
   (b) On both hands, using your finger bones for a base 12 system.
   (c) On one hand, using finger binary.
6. A line from entrepreneur Erlich Bachman in *Silicon Valley* (S1E1):

   > I memorized the hexadecimal times tables when I was fourteen writing machine code, okay? Ask me what 9 times F is. It's fleventy five!

   Please double-check Mr. Bachman's arithmetic.
7. Express the following numbers in IEEE double precision:
   a) ∞
   b) 17.25
   c) -33×2^{-22}
   d) 2^{-1036}
8. What numbers are represented by the following, assuming the IEEE double precision standard?
   a) `0 01111110001 1010` $\underbrace{\texttt{0...0}}_{49}$
   b) `0 10000000000 101010` $\underbrace{\texttt{0...0}}_{47}$
   c) `1 10100110000` $\underbrace{\texttt{0...0}}_{52}$
9. How many IEEE double-precision floating point numbers lie in the interval $[1,8]$? Describe the spacing between consecutive numbers in this range.
10. For which floating point numbers x is the statement "The smallest floating point number greater than x and the largest floating point number less than x are equally far away from x" false?
11. How does Matlab evaluate each of the following expressions? Make a prediction before you test each one. Did any of the results surprise you?

    a) `0/0`
    b) `1/0`
    c) `-1/0`
    d) `0*Inf`
    e) `Inf - Inf`
    f) `exp(-Inf)`

g) `cos(Inf)`
h) `1/-Inf`
i) `Inf/Inf`
j) `0*nan`
k) `0^0`
l) `Inf^0`

12. What is `eps(1e17)`? Can the integer $10^{17}+16$ be represented exactly as a double precision float? What about $10^{17}+8$? Or $10^{17}+32$? *Note*: The number 10^{17} can be represented exactly in double precision.
13. In order to evaluate the binomial coefficient $\binom{100}{40}$, we can use Matlab's `nchoosek(100,40)`. If we do, however, we get the warning `Warning: Result may not be exact. Coefficient is greater than 9.007199e+15 and is only accurate to 15 digits.` What is special about the number `9.007199e+15`? Why does Matlab give the warning here but not for e.g. `exp(60)`?
14. (Higham [10]) Show that

$$0.1 = \sum_{i=1}^{\infty} (2^{-4i} + 2^{-4i-1})$$

and deduce that 0.1 has the base 2 representation $0.000\overline{1100}$ (repeating last 4 bits). Let $\tilde{x} = \mathrm{fl}(0.1)$ be the rounded version of 0.1 in IEEE single precision arithmetic ($u = 2^{-24}$). Show that $(x-\tilde{x})/x = -\frac{1}{4}u$.
15. (Higham) Which of the following statements are true in IEEE arithmetic, assuming that a and b are normalized floating point numbers and that no overflow or underflow occurs?
    (a) $\mathrm{fl}(a+b) = \mathrm{fl}(b+a)$
    (b) $\mathrm{fl}(b-a) = -\mathrm{fl}(a-b)$
    (c) $\mathrm{fl}(a+a) = \mathrm{fl}(2*a)$
    (d) $\mathrm{fl}(0.5*a) = \mathrm{fl}(a/2)$
    (e) $\mathrm{fl}((a+b)+c) = \mathrm{fl}(a+(b+c))$
    (f) $a \le \mathrm{fl}((a+b)/2) \le b$, given that $a \le b$
16. (Higham) Barring overflow and underflow, are there any floating point numbers x and y such that the computed value of $x/\sqrt{x^2+y^2}$ is greater than 1? (This computation finds the angle of a triangle based on the lengths of its legs.)
17. Write two programs to compute $\sum_{n=1}^{100} n^3$ using each of the following methods:
    - Using a `for` loop.
    - Using array operations and the `sum` function, but no `for` loops.
18. (Taylor) An experimenter measures the separate masses M and m of a car and trailer. He gives his results in the standard form $M_{\text{best}} + \delta M$ and $m_{\text{best}} + \delta m$. What would be his best estimate for the total mass $M+m$? By considering the largest and smallest probable values of the total mass, show that his uncertainty in the total mass is just the sum of δM and δm. State your arguments clearly; don't just write down the answer.
19. Consider the function

$$f(x) = \frac{1}{1+\frac{1}{2+1/x}},$$

and the Matlab implementation `f = @(x)1./(1+1./(2+1./x))`.
    (a) For what real numbers x is $f(x)$ undefined? What does the Matlab function `f` return on those inputs?
    (b) In what way is Matlab's behavior reasonable? In what way is it problematic?
20. (Higham) Two requirements that we might ask of a routine for computing $\sqrt{x}$ in floating point arithmetic are that the identities $\sqrt{x^2} = |x|$ and $(\sqrt{x})^2 = |x|$ be satisfied. Which, if either, of these is a reasonable requirement?

2. Error Analysis

2.1 Conditioning and Stability

It is worth asking at this point whether rounding errors are a big deal. If numbers are stored to 16 decimal places, how much can a few rounding errors really affect the accuracy of our algorithms?

As a motivating example, we compare two algorithms for estimating the value of π using only elementary arithmetic operations. Start by inscribing a hexagon in a circle of radius 1. The hexagon has $e_1 = 6$ edges, each of length $s_1 = 1$, so the fact that the circle encloses the hexagon implies that $\pi \geq s_1 e_1/2 = 3$.

Now double the number of sides to get a 12-gon inscribed in the circle. It has $e_2 = 12$ edges, each of length $s_2 = \sqrt{2-\sqrt{3}} \approx 0.5176$, resulting in the improved approximation

$$\pi \geq s_2 e_2/2 \approx 3.1058.$$

Carrying on in this manner: for $n = 1, 2, \ldots$, let $e_n = 3 \cdot 2^n$ be the number of edges of the n-th polygon and let s_n be the length of each edge. By using the sine half-angle formula appropriately, it

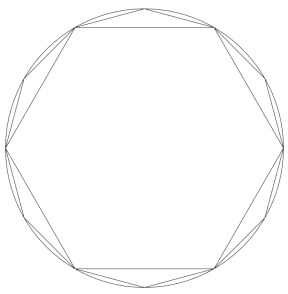


Figure 2.1: A hexagon and 12-gon inscribed in a circle

can be shown that the edge lengths satisfy the recurrence

$$s_{n+1} = \sqrt{2 - \sqrt{4 - s_n^2}}. \tag{2.1}$$

Program 2.1: Approximate the value of pi

```
s = 1;
e = 6;
for n = 1:30
    p(n) = s*e/2;
    e    = 2*e;
    s    = sqrt(2-sqrt(4-s^2));
end
```

In exact arithmetic, the limit $\lim_{n\to\infty} p_n = \pi$ will hold. In IEEE double precision arithmetic, however, the algorithm reaches its most accurate approximation at $n = 13$ with $p_{13} \approx$ 3.141592645321216 (correct digits underlined). The approximations begin to degrade shortly thereafter, eventually reaching the unfortunate result $p_{29} = 0$. Even though Program 2.1 uses only elementary arithmetic operations and square roots, it fails completely in under 30 iterations!

A second implementation makes a small modification to the recurrence. By taking Equation (2.1) and multiplying by the conjugate radical, it follows that

$$s_{n+1} = \sqrt{2 - \sqrt{4 - s_n^2}} \left(\frac{\sqrt{2 + \sqrt{4 - s_n^2}}}{\sqrt{2 + \sqrt{4 - s_n^2}}} \right) = \frac{\sqrt{4 - (4 - s_n^2)}}{\sqrt{2 + \sqrt{4 - s_n^2}}} = \frac{s_n}{\sqrt{2 + \sqrt{4 - s_n^2}}}. \tag{2.2}$$

These two formulas are equivalent in exact arithmetic, but by making this seemingly minor change, the resulting Program 2.2 approximates all bits of pi correctly by iteration 26! Results are shown in Figure 2.2, with correct digits underlined.

Program 2.2: Modified approximation for pi

```
s = 1;
e = 6;
for n = 1:30
    p(n) = s*e/2;
    e    = 2*e;
    s    = s/sqrt(2+sqrt(4-s^2));
end
```

What accounts for the difference between the two algorithms? We'll get to an explanation shortly, but for now will summarize the experiment with the following observation.

Fact 2.1.1 Algorithms can be equivalent in exact arithmetic but still behave very differently in finite precision.

2.1.1 Condition Numbers

For one reason or another, we might find ourselves wanting to compute $f(x)$ for some function f and input x, but instead we end up computing $f(\tilde{x})$ for some slightly different input $\tilde{x}$. If $\tilde{x}$ is close to x, how closely can we expect $f(\tilde{x})$ to approximate the desired output $f(x)$?

The answer depends on f and x. To

Definition 2.1.1 The **absolute condition number** of a function f at an input x is

$$\kappa_{\text{abs}} := \limsup_{\tilde{x}\to x} \frac{|f(\tilde{x}) - f(x)|}{|\tilde{x} - x|}.$$

Informally, if $\tilde{x}$ is "sufficiently close" to x then κ_{abs} will approximately satisfy the bound

$$|f(\tilde{x}) - f(x)| \lessapprox \kappa_{\text{abs}} |\tilde{x} - x|.$$

Definition 2.1.2 The **relative condition number** of a function f at an input x satisfying $x \neq 0$, $f(x) \neq 0$, is

$$\kappa_{\text{rel}} := \limsup_{\tilde{x}\to x} \frac{|f(\tilde{x}) - f(x)|/|f(x)|}{|\tilde{x} - x|/|x|}.$$

Informally, if $\tilde{x}$ is "sufficiently close" to x then κ_{rel} will approximately satisfy the bound

$$\frac{|f(\tilde{x}) - f(x)|}{|f(x)|} \lessapprox \kappa_{\text{rel}} \cdot \frac{|\tilde{x} - x|}{|x|}.$$

Intuitively, the condition number puts an upper bound on the ratio between the error in the input and the error in the output. If the relative condition number of f at x is about 10^d, we can expect to lose up to d digits of precision in the output $f(\tilde{x})$ relative to the error in the input $\tilde{x}$.

Fact 2.1.2 If f is a differentiable function, then for a given point x the absolute and relative condition numbers are given by

$$\kappa_{\text{abs}} = |f'(x)|,$$
$$\kappa_{\text{rel}} = \frac{|xf'(x)|}{|f(x)|}.$$

In this way, the absolute condition number is just a glorified term for the derivative, relying on the linear approximation

$$f(x+h) = f(x) + hf'(x) + \mathcal{O}(h^2).$$

The relative condition number is related, but takes into account the relative magnitudes of x and $f(x)$.

■ **Example 2.1** Let $f(x) = 1/x$ where $x \neq 0$. Then $\kappa_{\text{abs}} = -1/x^2$ and $\kappa_{\text{rel}} = 1$. The absolute condition number tells us that if x is small, a small *absolute* perturbation in the input can lead to a much larger change in the output. The relative condition number tells us that *regardless of the magnitude of* x, if we know x to k digits of accuracy then we can expect to compute $1/x$ to at least k digits of accuracy. ■

Definition 2.1.3 A problem is **well-conditioned** in the relative sense if κ_{rel} is small (i.e., not too much larger than 1). If κ_{rel} is large, the problem is **ill-conditioned** or **badly conditioned**.

As with errors and uncertainties, we typically care just about the order of magnitude of the condition number rather than its exact value.

Program 2.3: A poorly conditioned function

```
x  = vpa(1e8) + 2*pi; % exact input, using variable precision arithmetic
xt = 1e8 + 2*pi;      % approximate
kappa = double(abs(x*tan(x))); % relative condition number
errIn = double(abs((x-xt)/x));

y  = cos(x);
yt = cos(xt);
errOut = double(abs((y-yt)/y));
```

Rule of Thumb 2.1.3 Condition numbers should generally be reported to at most one significant digit.

■ **Example 2.2** Let $f(x) = \cos(x)$ and let $x = 10^8 + 2\pi$. Then $\kappa_{\text{abs}} \approx 0.93$, (the derivative of the cosine function is always bounded by 1), so the problem is very well conditioned in the absolute sense. But $\kappa_{\text{rel}} = |x\tan(x)| \approx 2.6 \cdot 10^8$, so the problem is badly conditioned in the relative sense.

To examine the consequence of having a poor condition number, we let $x = 10^8 + 2\pi$ and $\tilde{x} = \text{fl}(x)$ (since x cannot be represented in double precision). Due to rounding, the input has a relative error of about $4.0 \cdot 10^{-17}$. For the outputs, we get

$$\cos(x) = -0.363385089355691,$$
$$\cos(\tilde{x}) = -0.363385085658598,$$

for a relative error of about $1.0 \cdot 10^{-8}$. The output error is fairly close to κ_{rel} times the relative error in the input. If we don't know the relative error in the input ahead of time, we at least know in this case that it is bounded by the unit roundoff $\mathtt{u} \approx 1.11 \cdot 10^{-16}$, and so can get a relative error bound of $\kappa_{\text{rel}} \cdot \mathtt{u} \approx 2.8 \cdot 10^{-8}$. ■

What makes the cosine function so problematic here where the relative error is concerned? The function oscillates with a period of 2π. Since the magnitude of x is rather large, a small relative error in x translates to a significantly larger absolute error.

■ **Example 2.3** Let $f(x) = \log(x)$, and consider the same inputs $x = 10^8 + 2\pi$ and $\tilde{x} = \text{fl}(x)$ as before. This time, the relative condition number is $\kappa_{\text{rel}} = \log(x)^{-1} \approx 0.054$, so the relative error in the output will be *smaller* than the error in the input! As it turns out, $f(x)$ and $f(\tilde{x})$ agree to full double precision. ■

■ **Example 2.4** Let $f(x) = \alpha x$ for some constant $\alpha \neq 0$. Then the relative condition number is $\kappa_{\text{rel}} = 1$. Thus multiplication by a scalar is a well-conditioned operation, preserving but not amplifying the input error. ■

■ **Example 2.5 — Pathological Example.** Consider the function

$$f(x) = \begin{cases} 0 & \text{if } x \text{ is irrational,} \\ 1 & \text{if } x \text{ is rational.} \end{cases}$$

No perturbation to x, no matter how small, will permit us to guarantee that $f(\tilde{x})$ is close to $f(x)$. In this case, $\kappa_{\text{rel}} = \infty$ and $\kappa_{\text{abs}} = \infty$. ■

n	Program 2.1	Program 2.2
1	3.000000000000000	3.000000000000000
2	3.105828541230250	3.105828541230249
3	3.132628613281237	3.132628613281238
4	3.139350203046872	3.139350203046867
5	3.141031950890530	3.141031950890509
6	3.141452472285344	3.141452472285462
7	3.141557607911622	3.141557607911857
8	3.141583892148936	3.141583892148318
9	3.141590463236762	3.141590463228050
10	3.141592106043048	3.141592105999271
11	3.141592516588155	3.141592516692156
12	3.141592618640789	3.141592619365383
13	3.141592645321216	3.141592645033690
14	3.141592645321216	3.141592651450766
15	3.141592645321216	3.141592653055036
16	3.141592645321216	3.141592653456103
17	3.141593669849427	3.141592653556370
18	3.141592303811738	3.141592653581437
19	3.141608696224804	3.141592653587703
20	3.141586839655041	3.141592653589270
21	3.141674265021758	3.141592653589662
22	3.141674265021758	3.141592653589759
23	3.143072740170040	3.141592653589784
24	3.159806164941135	3.141592653589790
25	3.181980515339464	3.141592653589792
26	3.354101966249685	3.141592653589793
27	4.242640687119286	3.141592653589793
28	6.000000000000000	3.141592653589793
29	0.000000000000000	3.141592653589793
30	0.000000000000000	3.141592653589793

Figure 2.2: Results of two different methods for approximating pi

2.1.2 Catastrophic Cancellation

One common source of ill-conditioning comes from computing the difference $x-y$ of two numbers that are close to each other (or equivalently, computing $x+y$ when $y \approx -x$). If the difference is significantly smaller in magnitude than the original data, then any uncertainty in the input can result in a much greater relative uncertainty in the output.

■ **Example 2.6** Alvin and Belinda are both about six feet tall. How much taller is Belinda? Given the approximate measurements, the best we can say is that the two are about the same height. If the true answer is a quarter inch, then our answer has no relative accuracy at all. ■

This phenomenon does not depend on roundoff error or floating point arithmetic—all it requires is some uncertainty in the input, and it could just as easily come from measurement error, statistical noise, or any number of other sources. But when working with floating point arithmetic, roundoff error will often introduce a small amount of uncertainty in the computations.

■ **Example 2.7** Let $x = 10^{14} + 0.1$ and let $y = 10^{14}$. Then $x - y = 0.1$ but in IEEE double precision $\text{fl}(x-y) = 0.09375$, which has a relative error of about 0.06. The error came about because x cannot be stored exactly in double precision and so was rounded to a nearby number $\tilde{x} = \text{fl}(x)$. ■

■ **Example 2.8** Let $x = 10^{14} + 1$ and let $y = 10^{14}$. Then $\text{fl}(x-y) = x - y = 1$. There is no error despite the cancellation because the values of x and y are stored exactly. ■

In order to derive condition numbers for cancellation, we're going to need the *triangle inequality*, one of a numerical analyst's favorite tools. It allows us to derive error bounds by breaking a complicated expression into simpler ones.

Theorem 2.1.4 — Triangle Inequality. For any real numbers x and y, $|x \pm y| \leq |x| + |y|$.

More precisely, we will analyze the condition number of the function $f(x,y) = x - y$ (analysis for $f(x,y) = x+y$ will be almost identical). Definitions 2.1.1 and 2.1.2 only cover the case when f is a function of a single variable, so for the purposes of this section we will define the absolute input error as $\max\{|\tilde{x}-x|, |\tilde{y}-y|\}$ and the relative input error as $\max\{|\tilde{x}-x|/|x|, |\tilde{y}-y|/|y|\}$.

Lemma 2.1.5 — Subtraction is well-conditioned in the absolute sense. Let $\tilde{x} \approx x$ and $\tilde{y} \approx y$ be real numbers. The absolute error in $f(\tilde{x},\tilde{y}) = \tilde{x} - \tilde{y}$ is

$$|(\tilde{x}-\tilde{y}) - (x-y)| = |(\tilde{x}-x) - (\tilde{y}-y)| \overset{\triangle \text{ Ineq.}}{\leq} |\tilde{x}-x| + |\tilde{y}-y| \leq 2\max\{|\tilde{x}-x|, |\tilde{y}-y|\}.$$

Thus $\kappa_{\text{abs}} = 2$, and so subtraction is absolutely well-conditioned.

The error in the output will at least be small compared to the original inputs x and y. However, it will not necessarily be small compared to the true difference $|x-y|$.

Theorem 2.1.6 — Subtraction can be ill-conditioned in the relative sense. Let $\tilde{x} \approx x$ and $\tilde{y} \approx y$ be real numbers. The relative error in $f(\tilde{x},\tilde{y}) = \tilde{x} - \tilde{y}$ is

$$\begin{aligned} \frac{|(\tilde{x}-\tilde{y}) - (x-y)|}{|x-y|} \leq \frac{|\tilde{x}-x| + |\tilde{y}-y|}{|x-y|} &= \frac{1}{|x-y|}\left(|x| \cdot \frac{|\tilde{x}-x|}{|x|} + |y| \cdot \frac{|\tilde{y}-y|}{|y|}\right) \\ &\leq \frac{|x|+|y|}{|x-y|} \cdot \max\left\{\frac{|\tilde{x}-x|}{|x|}, \frac{|\tilde{y}-y|}{|y|}\right\}. \end{aligned}$$

Thus the relative condition number $\kappa_{\text{rel}} = (|x|+|y|)/|x-y|$ can be quite large if $|x-y|$ is small compared to $|x|+|y|$, as is the case in Example 2.7.

Fortunately, cancellation is the only situation for which an elementary arithmetic operation can be badly conditioned. Adding numbers of the same sign (or subtracting numbers with opposite signs), multiplication, division, and square roots are all very well-conditioned in the relative sense.

2.2 Avoiding Cancellation

When talking about error that arises in computation, it is useful to distinguish between a *function* and an *algorithm for evaluating a function*. Informally, we will say that a function f is a rule that maps elements from one set to another, while an algorithm f^* is a set of instructions executable by a computer. We can say that a function is well-conditioned or ill-conditioned, and similarly we can refer to an algorithm as being *stable* or *unstable*.

Definition 2.2.1 An algorithm f^* is **forward stable** if for all inputs x,

$$\frac{|f^*(x) - f(x)|}{|f(x)|} \leq C \cdot \kappa_{\text{rel}} \cdot \mathtt{u},$$

where $\mathtt{u}$ is the unit roundoff and C is "small" (close to 1, or at most a few orders of magnitude larger). Note that κ_{rel} may depend on x.

We might encounter one of several different situations:

- If the function is well-conditioned and our algorithm is stable, then everything is fine!
- If the function is ill-conditioned, then a small amount of uncertainty in the input may cause a large amount of uncertainty in the output. One technique for dealing this this problem is called *regularization*, where we evaluate a function that is closely related to the original one but has a better condition number. More on this topic later.
- If the function is well-conditioned but the algorithm is unstable, then we can hope to find a more stable algorithm.

In this section we discuss a few small tricks for dealing with the third case.

■ **Example 2.9** The function $f(x) = 3x$ is very well-conditioned, with $\kappa_{\text{rel}} = 1$. The algorithm f_1^* given by `f1 = @(x)3*x` is forward stable, since for $x \neq 0$ we have

$$\frac{|f_1^*(x) - f(x)|}{|f(x)|} = \frac{|3x(1+\delta) - 3x|}{|3x|} = |\delta| \leq u = \kappa_{\text{rel}} \cdot u.$$

Here we use Theorem 1.3.1 to bound the error from the multiplication operation $\text{fl}(3x)$. The algorithm f_2^* given by `f2 = @(x)(1e8+3)*x - (1e8)*x` is not forward stable, since for $x \neq 0$ we have

$$\begin{aligned} f_2^*(x) &= \text{fl}\left(\text{fl}((10^8+3)x) - \text{fl}(10^8 x)\right) \\ &= \left((10^8+3)x(1+\delta_1) - 10^8 x(1+\delta_2)\right)(1+\delta_3) \\ &= 3x(1+\delta_1)(1+\delta_3) + 10^8 x(\delta_1 - \delta_2)(1+\delta_3), \end{aligned}$$

and therefore

$$\begin{aligned} \frac{|f_2^*(x) - f(x)|}{|f(x)|} &\leq |(\delta_1 + \delta_3 + \delta_1\delta_3) + \frac{10^8}{3}(\delta_1 - \delta_2)(1+\delta_3)| \\ &\leq (2u + u^2) + \frac{10^8}{3}(2u)(1+u) \\ &\approx \frac{2 \cdot 10^8}{3} u. \end{aligned}$$

The inequality uses $|\delta| \leq u$ along with repeated use of the triangle inequality (2.1.4). Sure enough, testing out the functions in Matlab gives

$$\begin{aligned} 3\pi &= 9.4247779607693797\ldots, \\ f_1^*(\pi) &= 9.424777960769379, \\ f_2^*(\pi) &= \underline{9.424777}84619141. \end{aligned}$$

■

■ **Example 2.10** Consider the problem of evaluating

$$1-\cos(x), \quad \text{for} \quad x \approx 0.$$

The straightforward implementation will suffer from cancellation because $\cos(x) \approx 1$ when $x \approx 0$. However, the function is not itself ill-conditioned. Using the Taylor series

$$\cos(x) = 1 - \frac{x^2}{2} + \mathcal{O}(x^4),$$

we find that $1-\cos(x) \approx \frac{x^2}{2}$ for $x \approx 0$, and the function $f(x) = x^2/2$ has a relative condition number of 1. It turns out that we can find a more stable implementation by using a trig identity. Since $\sin^2(x)+\cos^2(x) = 1$, it follows that $\sin^2(x) = 1-\cos^2(x) = (1+\cos(x))(1-\cos(x))$, and therefore

$$1-\cos(x) = \frac{\sin(x)}{1+\cos(x)}.$$

When $x \approx 0$, the denominator will satisfy $1+\cos(x) \approx 2$ and will not suffer from cancellation. ■

2.2.1 Roots of the quadratic equation

Consider the quadratic equation $ax^2+bx+c = 0$, and the standard formula

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

If $|b^2| \gg |4ac|$, then the formula for one of the two roots (the root closer to zero) will suffer from cancellation.

■ **Example 2.11** If $a = c = 1$ and $b = -5600$, then compared with Matlab's `roots`, the larger root x_+ will be computed exactly by the above formula, but the smaller root x_- will have a relative error of about $7 \cdot 10^{-10}$. ■

However, consider the roots as functions of b: for fixed a and c, define

$$f_\pm(b) = \frac{-b \mp \text{sign}(b)\sqrt{b^2-4ac}}{2a},$$

where $\text{sign}(b) = 1$ for $b \geq 0$ and $\text{sign}(b) = -1$ for $b < 0$. Some algebra shows that the relative condition number of either function with respect to b is

$$\kappa_{\text{rel}} = \frac{|b|}{\sqrt{b^2-4ac}}.$$

So if $b^2-4ac \approx 0$ (i.e., the roots are very close together) the problem is ill-conditioned with respect to uncertainty in b, but when $|b| \gg |4ac|$ the relative condition number is close to 1! So the inaccuracy from our first attempt is due to a problem with our *algorithm* rather than the function itself. We instead note that if

$$ax^2+bx+c = a(x-x_+)(x-x_-),$$

then by equating the constant coefficients on both sides it follows that $x_+x_- = c/a$. We therefore can use the more stable algorithm

$$x_+ = \frac{-(b+\text{sign}(b)\sqrt{b^2-4ac})}{2a}, \quad x_- = \frac{c}{ax_+}.$$

The addition operation in the formula for x_+ avoids cancellation because the summands have the same sign, and the formula for x_- involves only a multiplication and a division. The cancellation in the term b^2-4ac, on the other hand, is unavoidable. When $b^2-4ac \approx 0$, the roots become more sensitive to small perturbations in b.

2.2.2 Complex step differentiation

Suppose we want to estimate $f'(x)$, where f is an analytic function and $f(x)$ is real. A common trick for approximating the derivative is to use the divided difference formula

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}.$$

One justification for this approximation relies on the Taylor series expansion of f: since

$$f(x+h) = f(x)+hf'(x)+\frac{h^2}{2}f''(x)+\mathcal{O}(h^3),$$
$$f(x+h) = f(x)-hf'(x)+\frac{h^2}{2}f''(x)+\mathcal{O}(h^3),$$

it follows that

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x)+\mathcal{O}(h^2).$$

So by making h small, we can get a good approximation of $f'(x)$.

Unfortunately, once we leave the world of exact arithmetic and try to use an actual machine, it becomes apparent that this trick comes with a catch. Suppose that due to rounding error we end up computing $\mathrm{fl}(f(x+h)) = f(x+h)(1+\delta_1)$ and $\mathrm{fl}(f(x-h)) = f(x-h)(1+\delta_2)$, where (optimistically) $|\delta_i| \le u$ for $i = 1,2$. Ignoring other sources of roundoff error for simplicity, we then end up with the approximation

$$\mathrm{fl}\left(\frac{f(x+h)-f(x-h)}{2h}\right) = f'(x)+f(x)\frac{(\delta_1-\delta_2)}{2h}+\mathcal{O}(u+h^2).$$

The smaller h gets, the larger the second term grows, and so by setting $h = 10^{-5}$ or so the best we can hope for is to get about 10 significant digits in our approximation. More elaborate finite difference methods can recover more digits, but are more expensive and still face a limit to how small h can get before the error increases.

So what if we take a step along the *imaginary* axis instead? We could expand $f(z)$ as the Taylor series

$$f(x+ih) = f(x)+ihf'(x)-\frac{h^2}{2}f''(x)+\mathcal{O}(h^3),$$

and by considering just the imaginary part conclude that

$$\frac{\mathrm{Im}(f(x+ih))}{h} = f'(x)+\mathcal{O}(h^2).$$

Almost magically, the cancellation that previously plagued us has vanished!

■ **Example 2.12** Take the function

$$f(x) = \frac{e^x}{(\cos(x))^3+(\sin(x))^3}$$

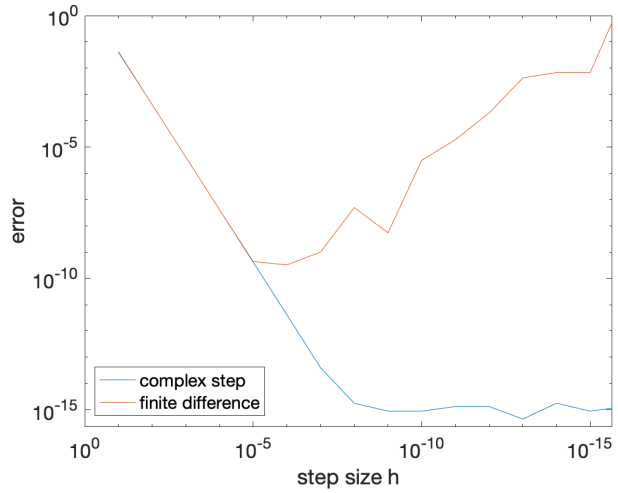


Figure 2.3: Complex step differentiation is accurate for small step sizes while the finite difference method never attains full accuracy.

and try to approximate its derivative at $x = \pi/4$. The exact derivative is $f'(x) = \sqrt{2}e^{\pi/4} \approx 3.10$. The code in Program 2.4 tests both complex step and finite difference methods for values of $h = 10^{-k}$, $1 \leq k \leq 16$. Results are shown in Figure 2.3: the finite difference method bottoms out at $h = 10^{-6}$ with an absolute error of about $3.26 \cdot 10^{-10}$, but the complex step method reaches full accuracy by $h = 10^{-8}$ and thereafter gives errors ranging from 2ε to 8ε. ∎

Program 2.4: Complex step vs finite difference method. Source: Cleve Moler[18]

```
syms x
F = exp(x)/((cos(x))^3 + (sin(x))^3);
exact = subs(diff(F),pi/4);
flexact = double(exact);

f = @(x)exp(x)./((cos(x)).^3+(sin(x)).^3);
fdc = @(x,h) imag(f(x+1i*h))/h;
fdr = @(x,h) (f(x+h) - f(x-h))/(2*h);

h = 10.^(-(1:16)');
errs = zeros(16,2);
for k = 1:16
  errs(k,1) = abs(fdc(pi/4,h(k))-flexact);
  errs(k,2) = abs(fdr(pi/4,h(k))-flexact);
end
loglog(h,errs)
axis([eps 1 eps 1])
set(gca,'xdir','rev','fontsize',16)
legend('complex step','finite difference','location','southwest')
xlabel('step size h')
ylabel('error')
```

So what made the complex step method work so well? One way of looking at it is that the terms $f(x)$ and $f''(x)$ in the Taylor series expansion have imaginary part *exactly* equal to zero, so there is no uncertainty to be amplified when we remove these terms and take the imaginary part of

Program 2.5: Recursive multiplication

```
function p = rec_mult(x)
    p = x(1)
    for i = 2:n
        p = p*x(i)
    end
end
```

$f(x+ih)$. There is one caveat: we need to be able to evaluate $f(x+ih)$ accurately, which may limit the range of functions f to which we can apply this technique.

2.3 Error Analysis

In general, if we want to use an algorithm we would like to have bounds on what level of accuracy we can expect from it. The most straightforward error bounds combine the rounding rule of Theorem 1.3.1 with the triangle inequality (Theorem 2.1.4). Here we present two examples, one for multiplication and one for summation.

2.3.1 Multiplication

Say we want to compute a product of floating point numbers $\prod_{i=1}^{n} x_i$. Since multiplication is defined as a *binary* operation, we can compute the overall product using *recursive multiplication*, shown in Program 2.5. For the analysis: using Theorem 1.3.1, we find that

$$\begin{aligned}
\tilde{p}_1 &= x_1 = p_1, \\
\tilde{p}_2 &= \mathrm{fl}(\tilde{p}_1 x_2) = p_1 x_2 (1+\delta_2) = p_2(1+\delta_2), \\
\tilde{p}_3 &= \mathrm{fl}(\tilde{p}_2 x_3) = (p_2(1+\delta_2)x_3)(1+\delta_3) = p_3(1+\delta_2)(1+\delta_3), \\
&\vdots \\
\tilde{p}_n &= \mathrm{fl}(\tilde{p}_{n-1} x_n) = p_n(1+\delta_2)\cdots(1+\delta_n).
\end{aligned}$$

The relative forward error can therefore be expressed as

$$\frac{|\tilde{p}_n - p_n|}{|p_n|} = |(1+\delta_2)\cdots(1+\delta_n) - 1|,$$

where $|\delta_i| \leq u$ for $i = 2:n$. Now clearly $(1-u)^n - 1 \leq (1+\delta_2)\cdots(1+\delta_n) - 1 \leq (1+u)^n - 1$, and it can be checked via calculus that $(1-u)^n - 1 \geq 1 - (1+u)^n$, so we can conclude that the relative forward error is bounded by

$$\frac{|\tilde{p}_n - p_n|}{|p_n|} \leq (1+u)^n - 1 = nu + \mathcal{O}(u^2).$$

Thus unless n is very large, we can expect recursive multiplication to have a high relative accuracy.

2.3.2 Summation

Say instead that we want to compute a product of floating point numbers $\sum_{i=1}^{n} x_i$, which we choose to do using *recursive summation* (Program 2.6).

The analysis proceeds as follows:

$$\begin{aligned}
\tilde{s}_1 &= x_1,\\
\tilde{s}_2 &= (\tilde{s}_1 + x_2)(1+\delta_2) = x_1(1+\delta_2) + x_2(1+\delta_2),\\
\tilde{s}_3 &= (\tilde{s}_2 + x_3)(1+\delta_3) = x_1(1+\delta_2)(1+\delta_3) + x_2(1+\delta_2)(1+\delta_3) + x_3(1+\delta_3),\\
&\vdots,\\
\tilde{s}_n &= \sum_{i=1}^{n} x_i \prod_{j=\max\{2,i\}}^{n} (1+\delta_j).
\end{aligned}$$

Program 2.6: Recursive summation

```
function s = rec_sum(x)
    s = x(1)
    for i = 2:n
        s = s + x(i)
    end
end
```

By the analysis from the section on multiplication and using the triangle inequality, we find that the relative forward error is bounded by

$$\frac{|\tilde{s}_n - s_n|}{|s_n|} \leq ((1+u)^n - 1)\frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|} = nu\frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|} + \mathcal{O}(u^2).$$

If there is no cancellation, then $\sum_{i=1}^n |x_i| = |\sum_{i=1}^n x_i|$ and the relative error will be small, growing proportionally to n. If there is lots of cancellation and $|\sum_{i=1}^n x_i| \ll \sum_{i=1}^n |x_i|$, then the problem is ill-conditioned in the relative sense.

2.4 Matlab Commands

Polynomials

- `roots(p)`: Computes the roots of the polynomial whose coefficients are elements of the vector **p**.
- `poly(r)`: Returns a vector whose elements are the coefficients of the polynomial whose roots are the elements of the vector **r**.
- `conv(p,q)`: Multiplies the polynomials with coefficients given by **p** and **q**.

Accurate computation

- `cospi(x)`: Computes $\cos(\pi x)$ accurately. See also `sinpi(x)`.
- `log1p(x)`: Computes $\log(1+x)$ accurately. Useful when x is near zero.
- `expm1(x)`: Computes $\exp(x)-1$ accurately. Useful when x is near zero.
- `hypot(x,y)`: Computes $\sqrt{x^2+y^2}$ carefully to avoid overflow or underflow.

Variable precision

- `S = sym(X)`: Constructs a symbolic variable `S` from `X`.
- `digits(D)`: Performs future calculations to `D` digits.
- `vpa(S)`: Evaluates `S` to a precision determined by the current value of `digits`.
- `vpa(S,D)`: Evaluates `S` to `D` digits instead of the current value of `digits`.

2.5 Exercises

1. Here you will look at the error analysis for a sum in MA 402 Tiny precision.
   (a) Evaluate the sum $9/8+2/5$ by hand in Tiny precision, making sure to round correctly after each binary operation.
   (b) What is the value of the unit roundoff u in Tiny precision?
   (c) Using Theorem 1.3.1 and the triangle inequality, derive an upper bound on the error in your computed answer. How does it compare to the actual error?
2. (Taylor) A student measures two numbers x and y as

   $$x = 10 \pm 1 \quad \text{and} \quad y = 20 \pm 1.$$

   What is her best estimate for their product $q = xy$? Using the largest probable values for x and y, calculate the largest probable value of q. Similarly, find the smallest probable value of q, and hence the range in which q probably lies.
3. You have a cylinder exactly 10cm tall. You estimate the circumference to be 8cm, plus or minus 1cm. (a) What should your estimate for the radius of the cylinder be, and what is the uncertainty in this estimate? (b) What about the volume of the cylinder? Use condition numbers to determine your answers. How do they compare with using the largest and smallest "probable values" for the circumference?
4. Suppose you have a perfect sphere. If you want to estimate the volume of the sphere to within 1 percent of its true value, how accurately must you measure the circumference?
5. Find the absolute and relative condition numbers of the following functions. For which values of x is the function especially ill-conditioned? Are there any values for which it is very well-conditioned?
   (a) $f(x) = \sqrt{1-x^2}$ for $x \in (-1,1)$
   (b) $f(x) = 1/(1+x)$ for $x \neq -1$
   (c) $f(x) = \sqrt{1+x^2}$
6. Find numbers x and y for which Matlab's `hypot(x,y)` gives an accurate answer but `sqrt(x^2+y^2)` does not.
7. If you type `cos(1e10/7)` into Matlab, about how many correct digits can you expect the answer to have?
8. (Taylor) With a good stopwatch and some practice, you can measure times ranging from approximately 1 second up to many minutes with an uncertainty of 0.1 second or so. Suppose that we wish to find the period τ of a pendulum with $\tau \approx 0.5$ s. If we time 1 oscillation, we have an uncertainty of approximately 20%; but by timing several oscillations together, we can do much better, as the following questions illustrate:
   (a) If we measure the total time for 5 oscillations and get 2.4 ± 0.1 s, what is our final answer for τ, with its absolute and percent uncertainties?
   (b) What if we measure 20 oscillations and get 9.4 ± 0.1 s?
   (c) Could the uncertainty in τ be improved indefinitely be timing more oscillations?
9. Consider the roots of the quadratic equation $ax^2+bx+c=0$ as functions of c, with a and b fixed. Find the absolute condition numbers with respect to c.
10. Consider the quadratic equation $10^{-5}x^2+10^{10}x+10^{-5}=0$. William Kahan [12] makes the following remark:

    > If we use the quadratic formula, we get roots of 2×10^{15} and 0. But zero is clearly not a root, and is accurate to no figures. Clearly, "cancellation is to blame." But, the subtraction was done with no error. The error was made when we did not carry thirty figures earlier. Cancellation does not cause error, but reveals earlier errors.

Which subtraction is Kahan referring to? What does he mean by the final statement?

3. Vectors, Matrices, Norms

3.1 Vectors

The space of n-dimensional column vectors with real elements is $\mathbb{R}^n$. A vector $\mathbf{x} \in \mathbb{R}^n$ has elements $x_j \in \mathbb{R}$, and is written as

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we say that $\mathbf{x} = \mathbf{y}$ if all corresponding elements of $\mathbf{x}$ and $\mathbf{y}$ are equal. Otherwise, we say $\mathbf{x} \neq \mathbf{y}$.

3.1.1 Special Vectors

The n-dimensional vectors of all zeros and all ones are, respectively,

$$\mathbf{0} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbb{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}.$$

The dimension n can be determined from the context.

Definition 3.1.1 — Standard basis vectors. The standard basis vectors $\mathbf{e}_j \in \mathbb{R}^n$ have a 1 in position j and 0 everywhere else.

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots, \quad \mathbf{e}_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

3.2 Matrices

The space of $m \times n$ matrices with real elements is $\mathbb{R}^{m\times n}$. A matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ has m rows and n columns with elements $a_{ij} \in \mathbb{R}$,

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

If we denote the jth column of $\mathbf{A}$ by $\mathbf{a}_j \in \mathbb{R}^m$, then we can write

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

■ **Example 3.1**

$$\mathbf{A} = \begin{bmatrix} 17 & 0 \\ \pi & -1.2 \\ 7 & 5 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{bmatrix} \quad \text{where} \quad \mathbf{a}_1 = \begin{bmatrix} 17 \\ \pi \\ 7 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} 0 \\ -1.2 \\ 5 \end{bmatrix}.$$

In Matlab: `A = [17,0; pi,-1.2; 7,5]` ■

3.2.1 Special Matrices

The all-zero and all-ones matrices in $\mathbb{R}^{m\times n}$ will be denoted by $\mathbf{0}_{m\times n}$ and $\mathbb{1}_{m\times n}$, respectively. The subscript will be omitted if the dimensions are clear from context.

The identity matrix $\mathbf{I}_n \in \mathbb{R}^{n\times n}$ has ones on the main diagonal and zeros everywhere else,

$$\mathbf{I}_n = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}.$$

The columns of $\mathbf{I}_n$ are the standard basis vectors in $\mathbb{R}^n$,

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

We again omit the subscript when the dimension is clear from context.

3.2.2 Notation

We generally use the following notation for matrices, vectors, and scalars:

- Matrices: uppercase boldface Roman or Greek letters: $\mathbf{A}$, Σ, etc.
- Vectors: lowercase boldface Roman letters: $\mathbf{x}$, $\mathbf{y}$.
- Scalars: lowercase Greek letters, e.g. α,
  or lowercase Roman letters with subscripts, e.g. x_i, a_{ij}.
- Integers for indexing and dimensions: i, j, k, ℓ, m, and n.

The special vectors/matrices $\mathbf{0}$ and $\mathbb{1}$ are exceptions.

3.3 Basic Operations

3.3.1 Transpose

The transpose of a vector or matrix is denoted $\mathbf{x}^T$ or $\mathbf{A}^T$. The transpose of a column vector is a row vector with the same elements. If $\mathbf{A} \in \mathbb{R}^{m\times n}$, then $\mathbf{A}^T \in \mathbb{R}^{n\times n}$ with

$$\mathbf{A}^T = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix}^T = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix}.$$

That is, the columns of $\mathbf{A}$ are the rows of $\mathbf{A}^T$ (and vice versa).

Fact 3.3.1 For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $(\mathbf{A}^T)^T = \mathbf{A}$.

Definition 3.3.1 A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is *symmetric* if $\mathbf{A}^T = \mathbf{A}$.

3.3.2 Inner Products

For any two column vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, the *inner product* or *dot product* is

$$\mathbf{x} \bullet \mathbf{y} = \mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i.$$

In Matlab: `x'*y` or `dot(x,y)`.

3.3.3 Matrix-Vector Products

The product of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with a vector $\mathbf{x} \in \mathbb{R}^n$ produces a vector $\mathbf{Ax} \in \mathbb{R}^m$. We can look at matrix-vector multiplication in at least two ways:

1. Element-wise:
   Element i of $\mathbf{Ax}$ is the inner product of row i of $\mathbf{A}$ with the vector $\mathbf{x}$,

$$(\mathbf{Ax})_i = \sum_{j=1}^{n} a_{ij}x_j = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad 1 \le i \le m.$$

2. Column-wise:
   $\mathbf{Ax}$ is a linear combination of columns of $\mathbf{A}$,

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

■ **Example 3.2** Any $\mathbf{x} \in \mathbb{R}^n$ is a linear combination of canonical basis vectors:

$$\mathbf{x} = \mathbf{Ix} = \mathbf{e}_1 x_1 + \cdots + \mathbf{e}_n x_n.$$

■

3.4 Vector Norms

Previously we measured the absolute error for scalars as $|\tilde{x} - x|$. If we have vectors $\tilde{\mathbf{x}}$ and $\mathbf{x}$ instead, we could get a vector of errors $|\tilde{\mathbf{x}} - \mathbf{x}|$. But if we want to express the "overall" error as a single number, how should we go about it? To answer this problem we introduce vector norms, which allow us to measure the "length" of a vector.

Definition 3.4.1 A vector norm $\|\cdot\|$ is a function from $\mathbb{R}^n$ to $\mathbb{R}$ satisfying three properties:

1. (Point-separating) If $\|\mathbf{x}\| = 0$ then $\mathbf{x} = \mathbf{0}$.
2. (Absolute homogeneity) For any $\mathbf{x} \in \mathbb{R}^n$ and any $\alpha \in \mathbb{R}$, $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$.
3. (Triangle inequality) For any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\|\mathbf{x}+\mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$.

R The first condition can equivalently be replaced by the condition

1*. (Positive definite) $\|\mathbf{x}\| \ge \mathbf{0}$ for all $\mathbf{x} \in \mathbb{R}^n$, and $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$.

Intuitively, this means that vectors cannot have negative length, and only the zero vector has zero length.

R Take care to remember the absolute values in the "absolute homogeneity" property. If one location is 40 miles west of where I am standing, and another is 40 miles east, then both are +40 miles away from me. That is, $\|-\mathbf{x}\| = \|\mathbf{x}\|$, and both quantities are nonnegative.

Fact 3.4.1 The triangle property in Definition 3.4.1 also implies the following properties:

- General triangle inequality: $\|\mathbf{x} \pm \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.
- Reverse triangle inequality: $|\|\mathbf{x}\| - \|\mathbf{y}\|| \leq \|\mathbf{x} \pm \mathbf{y}\|$.

■ **Example 3.3** For $\mathbf{x} \in \mathbb{R}^2$, the function $f(\mathbf{x}) = 2|x_1| + 5|x_2|$ is a norm. ■

■ **Example 3.4** For $\mathbf{x} \in \mathbb{R}^2$, the function $f(\mathbf{x}) = \min\{|x_1|, |x_2|\} + |x_1| + |x_2|$ is not a norm. It is point-separating and absolutely homogeneous, but does not satisfy the triangle inequality. Take $\mathbf{x} = [1,0]^T$ and $\mathbf{y} = [0,1]^T$ for a counterexample. ■

3.4.1 Our Favorite Vector Norms

One of the simplest types of vector norms to define is the p-norm.

Definition 3.4.2 For real numbers $p \geq 1$ and $\mathbf{x} \in \mathbb{R}^n$, the vector p-norm is

$$\|\mathbf{x}\|_p = \left(\sum_{j=1}^{n} |x_j|^p \right)^{1/p}.$$

The following three are some of the most commonly used for their simplicity:

- The 1-norm $\|\mathbf{x}\|_1 = \sum_{j=1}^{n} |x_j|$. Also known as the Manhattan or taxicab norm because it can represent your distance from a location when you are forced to walk on the sidewalks of a rectangular grid.
- The 2-norm or Euclidean norm $\|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{n} x_j^2} = \sqrt{\mathbf{x} \bullet \mathbf{x}}$. Represents our conventional notion of Euclidean distance.
- The infinity norm $\|\mathbf{x}\|_\infty = \max_{1 \leq j \leq n} |x_j|$. The norm is equal to the largest magnitude of any single element, in a "worst-case scenario" approach.

Because of the 2-norm's relation to inner products, it carries a number of attractive features that other norms do not.

Theorem 3.4.2 — Pythagorean theorem. If $\mathbf{x} \bullet \mathbf{y} = 0$, then $\|\mathbf{x} \pm \mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2$.

Inner products also satisfy a few useful inequalities in relation to norms. One famous one is the Cauchy-Schwarz inequality:

Theorem 3.4.3 — Cauchy-Schwarz inequality. For any vectors $\mathbf{x}$ and $\mathbf{y} \in \mathbb{R}^n$,

$$|\mathbf{x} \bullet \mathbf{y}| \leq \|\mathbf{x}\|_2 \|\mathbf{y}\|_2.$$

One generalization is known as Hölder's inequality:

Theorem 3.4.4 — Hölder's inequality. Let $p, q \in [1, \infty]$ satisfy $1/p + 1/q = 1$. Then for any $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$,

$$|\mathbf{x} \bullet \mathbf{y}| \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q.$$

■ **Example 3.5** Consider the three vectors

$$\mathbf{x}_1 = \begin{bmatrix} 5 \\ 6 \\ 6 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} 9 \\ 0 \\ 0 \end{bmatrix}.$$

Which of these has the largest magnitude? It depends on which norm we use.

	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_1$
$\mathbf{x}_1$	6	$\sqrt{86}$	**16**
$\mathbf{x}_2$	7	$\sqrt{\mathbf{99}}$	15
$\mathbf{x}_3$	**9**	9	9

Which of these norms should we use to compare the vectors, then? It depends on the application. ■

3.4.2 Consistency of Norms

The following theorem says that any two norms on finite-dimensional spaces can only differ from each other up to certain constants.

Theorem 3.4.5 For any two norms $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ on a finite-dimensional vector space $\mathscr{X}$, there exist positive constants c and C such that, for all $\mathbf{x} \in \mathscr{X}$,

$$c\|\mathbf{x}\|_\beta \leq \|\mathbf{x}\|_\alpha \leq C\|\mathbf{x}\|_\beta.$$

In the case of vector norms on $\mathbb{R}^n$, the constants might depend on the dimension n.

■ **Example 3.6** The following inequalities hold for all $\mathbf{x} \in \mathbb{R}^n$:

- $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq n\|\mathbf{x}\|_\infty$
- $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{x}\|_\infty$
- $\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2$

All bounds are tight. ■

Definition 3.4.3 Given a norm on $\mathbb{R}^n$, a *unit vector* is a vector $\mathbf{x} \in \mathbb{R}^n$ satisfying $\|\mathbf{x}\| = 1$. Typically, the 2-norm is used. The *unit sphere* is the set of points for which $\|\mathbf{x}\| = 1$, and the *unit ball* is the set of points for which $\|\mathbf{x}\| \leq 1$.

Figure 3.1 presents the unit spheres for our three favorite norms, showing that in $\mathbb{R}^2$ they agree with each other up to a factor of at most 2. They coincide at the standard basis vector $\mathbf{e}_j$, $j = 1:2$, and disagree the most for vectors with all entries equal to ± 1 —the corners of the unit square (or in n dimensions, the unit hypercube).

3.5 Matrix Norms

Similar to how vector norms can define the length of a vector, we can define matrix norms in order to quantify the "magnitude" of a matrix.

Definition 3.5.1 A matrix norm $\|\cdot\|$ is a function from $\mathbb{R}^{m\times n}$ to $\mathbb{R}$ satisfying three properties:

1. (Point-separating) If $\|\mathbf{A}\| = 0$ then $\mathbf{A} = \mathbf{0}$.
2. (Absolute homogeneity) For any $\mathbf{A} \in \mathbb{R}^{m\times n}$ and any $\alpha \in \mathbb{R}$, $\|\alpha\mathbf{A}\| = |\alpha|\|\mathbf{A}\|$.
3. (Triangle inequality) For any $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$, $\|\mathbf{A}+\mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$.

As with Definition 3.4.1, we can equivalently replace the first condition with the requirement

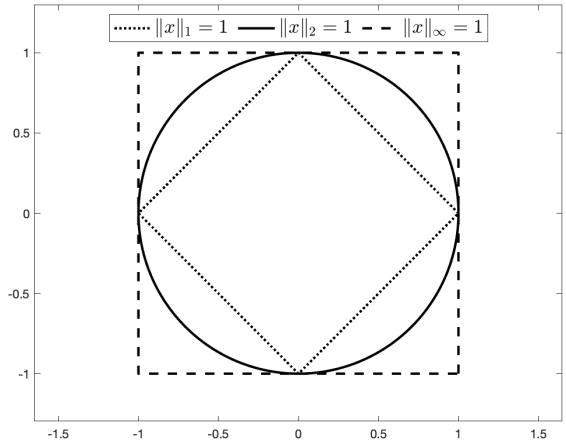


Figure 3.1: Unit spheres for the 1-norm, 2-norm and infinity norm.

that the norm be positive definite:

1*. (Positive definite) $\|\mathbf{A}\| \geq \mathbf{0}$ for all $\mathbf{A} \in \mathbb{R}^{m\times n}$, and $\|\mathbf{A}\| = 0$ if and only if $\mathbf{A} = \mathbf{0}$.

Depending on who you ask, matrix norms are sometimes also defined to be submultiplicative (a very useful property when dealing with products of matrices).

Definition 3.5.2 — Submultiplicativity. A matrix norm $\|\cdot\|$ is *submultiplicative* if for all matrices $\mathbf{A}$ and $\mathbf{B}$ whose product is well-defined, $\|\mathbf{AB}\| \leq \|\mathbf{AB}\|$.

3.5.1 Our Favorite Matrix Norms

One of our favorite types of matrix norms is defined in relation to a vector norm (or pair of vector norms).

Definition 3.5.3 Let $\|\cdot\|_\alpha$ be a vector norm on $\mathbb{R}^n$ and $\|\cdot\|_\beta$ be a vector norm on $\mathbb{R}^m$. Then the *induced norm* $\|\cdot\|_{\alpha,\beta}$ on $\mathbb{R}^{m\times n}$ is defined by

$$\|\mathbf{A}\|_{\alpha,\beta} = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{Ax}\|_\beta}{\|\mathbf{x}\|_\alpha}.$$

In particular, for the p-norms the corresponding induced norm is

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

By definition, these norms satisfy the bound

$$\|\mathbf{Ax}\|_\beta \leq \|\mathbf{A}\|_{\alpha,\beta}\|\mathbf{x}\|_\alpha.$$

We especially like the operator norm

$$\|\mathbf{A}\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\lambda_{\max}(\mathbf{A}^T\mathbf{A})},$$

but the 1-norm and infinity norm are useful in part because they are very simple to compute:

$$\|\mathbf{A}\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}| = \max_{1\le j\le n} \|\mathbf{a}_j\|_1, \tag{3.1}$$

$$\|\mathbf{A}\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}| = \max_{1\le i\le m} \|\mathbf{r}_i\|_1. \tag{3.2}$$

Here, $\mathbf{r}_1, \ldots, \mathbf{r}_m$ and $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are the rows and columns of $\mathbf{A}$, respectively.

Another popular norm, also simple to compute, is the Frobenius norm:

Definition 3.5.4 The *Frobenius norm* of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}$$

Fact 3.5.1 The p-norm is submultiplicative, as is the Frobenius norm.

3.6 Distances and Errors

Now that we have defined norms for vectors, we are prepared to define absolute and relative errors.

Definition 3.6.1 For vectors $\tilde{\mathbf{x}}$ and $\mathbf{x}$ and a norm $\|\cdot\|$, the absolute error is defined as $\|\tilde{\mathbf{x}} - \mathbf{x}\|$ and the normwise relative error is defined as $\|\tilde{\mathbf{x}} - \mathbf{x}\|/\|\mathbf{x}\|$.

■ **Example 3.7** Let $\mathbf{x} = [2,1,0]^T$ and $\tilde{\mathbf{x}} = [3,0,2]^T$. Then the absolute error is

$$\|\tilde{\mathbf{x}} - \mathbf{x}\|_2 = \|[1,-1,2]\|_2 = \sqrt{6},$$

and the relative error is

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \frac{\sqrt{6}}{\sqrt{5}}.$$

■

R The normwise relative error is *not* the same thing as defining a vector $\mathbf{y}$ by $y_i = \frac{\tilde{x}_i - x_i}{x_i}$ and returning $\|\mathbf{y}\|$. This measurement is different, but still might be useful depending on the circumstances.

More generally, the idea is that any norm $\|\cdot\|$ can be used to define a distance function by defining

$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|.$$

Definition 3.6.2 A *distance function* on a set $\mathscr{X}$ satisfies the properties for all $x, y, z \in \mathscr{X}$:

1. (Point separation) If $d(x,y) = 0$ then $x = y$
2. (Symmetry) $d(x,y) = d(y,x)$
3. (Triangle inequality) $d(x,y) \le d(x,z) + d(y,z)$

Necessarily, $d(x,y) \ge 0$ for all $x, y \in \mathscr{X}$. Given the relation between distance functions and norms, the distance function induced by a norm also satisfies $\|\mathbf{x}\| = d(\mathbf{x},\mathbf{0})$.

3.7 Condition Numbers for Multivariate Functions

Finally, we can rigorously extend the notion of absolute and relative condition numbers (Definitions 2.1.1 and 2.1.2) to higher-dimensional spaces.

Definition 3.7.1 Let f be a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. Given norms $\|\cdot\|_\alpha$ on $\mathbb{R}^m$ and $\|\cdot\|_\beta$ on $\mathbb{R}^n$, the **absolute condition number** of f at an input $x \in \mathbb{R}^n$ is

$$\kappa_{\text{abs}} := \limsup_{\tilde{\mathbf{x}} \to \mathbf{x}} \frac{\|f(\tilde{\mathbf{x}}) - f(\mathbf{x})\|_\beta}{\|\tilde{\mathbf{x}} - \mathbf{x}\|_\alpha}.$$

Informally, if $\tilde{\mathbf{x}}$ is "sufficiently close" to $\mathbf{x}$ then κ_{abs} will approximately satisfy the bound

$$\|f(\tilde{\mathbf{x}}) - f(\mathbf{x})\|_\beta \lessapprox \kappa_{\text{abs}} \|\tilde{\mathbf{x}} - \mathbf{x}\|_\alpha.$$

Definition 3.7.2 Let f be a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. Given norms $\|\cdot\|_\alpha$ on $\mathbb{R}^m$ and $\|\cdot\|_\beta$ on $\mathbb{R}^n$, the **relative condition number** of f at an input $\mathbf{x}$ satisfying $\mathbf{x} \neq 0$, $f(\mathbf{x}) \neq 0$, is

$$\kappa_{\text{rel}} := \limsup_{\tilde{\mathbf{x}} \to \mathbf{x}} \frac{\|f(\tilde{\mathbf{x}}) - f(\mathbf{x})\|_\beta / \|f(\mathbf{x})\|_\beta}{\|\tilde{\mathbf{x}} - \mathbf{x}\|_\alpha / \|\mathbf{x}\|_\alpha}.$$

Informally, if $\tilde{\mathbf{x}}$ is "sufficiently close" to $\mathbf{x}$ then κ_{rel} will approximately satisfy the bound

$$\frac{\|f(\tilde{\mathbf{x}}) - f(\mathbf{x})\|_\beta}{\|f(\mathbf{x})\|_\beta} \lessapprox \kappa_{\text{rel}} \cdot \frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|_\alpha}{\|\mathbf{x}\|_\alpha}.$$

3.7.1 Formula for Differentiable Functions

This section extends Fact 2.1.2 to the multivariate case. Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a differentiable function

$$f(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix},$$

and define the matrix of partial derivatives (i.e., the Jacobian)

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{bmatrix}.$$

From taking the partial derivatives of f at $\mathbf{x}$, we get the linear approximation

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + J(\mathbf{x})\mathbf{h} + \mathcal{O}(\|\mathbf{h}\|^2).$$

Consequently, by choosing $\mathbf{h}$ in the direction that maximizes $\|J(\mathbf{x})\mathbf{h}\|_\beta / \|\mathbf{h}\|_\alpha$, we find that

$$\begin{aligned} \kappa_{\text{abs}} &= \|J(\mathbf{x})\|_{\alpha,\beta}, \\ \kappa_{\text{rel}} &= \frac{\|\mathbf{x}\|_\alpha \|J(\mathbf{x})\|_{\alpha,\beta}}{\|f(\mathbf{x})\|_\beta}, \end{aligned}$$

where $\|\cdot\|_{\alpha,\beta}$ is the induced norm defined in Definition 3.5.3.

■ **Example 3.8** Consider the problem of computing a sum of numbers $f(\mathbf{x}) = \sum_{i=1}^{n} x_i$ discussed in Section 2.3.2. The matrix of partial derivatives is

$$J(\mathbf{x}) = \mathbb{1}^T = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix},$$

so if we take the condition number with respect to the 1-norm ($\|\cdot\|_\alpha = \|\cdot\|_1$), we find that

$$\begin{aligned} \|\mathbf{x}\|_1 &= \sum_{i=1}^{n} |x_i|, \\ \|J(\mathbf{x})\|_1 &= 1, \\ \|f(\mathbf{x})\|_1 &= \left| \sum_{i=1}^{n} x_i \right|, \end{aligned}$$

and therefore

$$\kappa_{\text{rel}} = \frac{\sum_{i=1}^{n} |x_i|}{\left|\sum_{i=1}^{n} x_i\right|},$$

where we determined $\|J(\mathbf{x})\|_1$ by using (3.1).

Therefore, turning back to the error analysis of Section 2.3.2, we conclude that the recursive summation algorithm of Program 2.6 satisfies

$$\frac{|f^*(\mathbf{x}) - f(\mathbf{x})|}{|f(\mathbf{x})|} \leq n \cdot \kappa_{\text{rel}} \cdot u + \mathcal{O}(u^2),$$

and is therefore forward stable as long as n is not too large. ■

3.8 Matlab Commands

Vectors

- `[x1;x2;...;xn]`: A column vector. See also `vertcat`.
- `[x1,x2,...,xn]`: A row vector. See also `horzcat`.
- `j:k`: The row vector `[j,j+1,...,j+m]` where `m=fix(k-j)`. Equal to `[j,j+1,...,k]` if `j` and `k` are both integers. Empty if `j>k`.
- `j:i:k`: The row vector `[j,j+i,...,j+m*i]` where `m=fix((k-j)/i)`. Negative values of `i` are permitted. Empty if `i==0`, `i>0` and `j>k`, or if `i<0` and `j<k`.
- `a(end)`: The final element of the vector `a`.
- `zeros(1,N)`: A length-N row vector of all zeros.
- `ones(1,N)`: A length-N row vector of all ones.
- `length(a)`: The length of the vector `a`.
- `linspace(x,y,N)`: A row vector with `N` linearly spaced points between `x` and `y`. `linspace(x,y)` uses $N = 100$ by default.
- `logspace(x,y,N)`: A row vector with `N` logarithmically spaced points between 10^x and 10^y. `logspace(x,y)` uses $N = 50$ by default.
- `a'`: The transpose of the vector `a`.

Logicals and Sorting

- `all(a)`: Returns `true` if all elements of the vector are `true`.
- `any(a)`: Returns `true` if any element of the vector is `true`.
- `find(a)`: Where `a` is a logical vector, returns the *indices* for which `a` is `true`.

- `max(a)`, `min(a)`: Find the maximum or minimum element of a vector. Using `[m,ix] = max(a)` will store the maximum element in `m` and its index in `ix`. See also `maxk` and `mink` for the k largest/smallest elements.
- `sort(a)`: Sorts the elements of `a`. Use 'ascend' (default) or 'descend' to control the sort order. Using `[s,ix] = sort(a)` returns the sorted vector `s` and the indices `ix` such that `a(ix) = s`.
- `issorted(a)`: Determines whether the elements of `a` are in sorted order.
- `median(a)`: Returns the median element of `a`.
- `a==b`: Element-wise comparison of elements in `a` and `b`.
- `isequal(a,b)`: Returns `true` if all elements of `a` and `b` are the same size and contain the same values, and `false` otherwise.

Arithmetic and Vectorization

- `sum(a)`: Returns the sum of the elements in `a`.
- `cumsum(a)`: Returns the cumulative sums of the elements in `a`.
- `prod(a)`: Returns the product of the elements in `a`.
- `cumprod(a)`: Returns the cumulative products of the elements in `a`.
- `diff(a)`: Returns the difference of consecutive elements in `a`. This will be a vector with length equal to `length(a) - 1`.
- `norm(a)`: Returns the 2-norm (Euclidean norm) of `a`. Use `norm(a,1)` or `norm(a,Inf)` for the 1-norm and infinity norm.
- `dot(a,b)`: Dot product of vectors `a` and `b`.
- `a.*b`: Elementwise product of vectors `a` and `b`. Same as `times(a,b)`.
- `a./b`: Elementwise division of vectors `a` and `b`. Same as `rdivide(a,b)`.
- `a.^x`: Elementwise powers of vector `a`. Same as `power(a,x)`.

Matrices

- `[c1,c2,...,cn]`: A matrix with column vectors $c_1, \ldots, c_n$ as its columns.
- `[r1;r2;...;rn]`: A matrix with row vectors $r_1, \ldots, r_n$ as its rows.
- `zeros(m,n)`: The $m \times n$ all-zero matrix. With only one argument, `zeros(m)` is the square $m \times m$ all-zero matrix.
- `ones(m,n)`: The $m \times n$ all-ones matrix. With only one argument, `ones(m)` is the square $m \times m$ all-ones matrix.
- `eye(m)`: The $m \times m$ identity matrix. More generally, `eye(m,n)` is an $m \times n$ matrix with ones on the main diagonal and zeros elsewhere.
- `A'`: Takes the transpose of `A`, or the conjugate transpose if `A` is complex. See also `transpose` and `ctranspose`.
- `size(A)`: Returns the number of rows and columns of `A`.
- `diag(A)`: A column vector with entries equal to the diagonal elements of `A`. `diag(A,k)` finds the k-th diagonal above ($k > 0$) or below ($k < 0$) the main diagonal.
- `diag(v)`: If `v` is a vector, creates a diagonal matrix from the entries of `v`. `diag(v,k)` puts the entries of `v` on the k-th diagonal above ($k > 0$) or below ($k < 0$) the main diagonal.
- `tril(A)`: Extracts the lower triangular part of `A` (all entries below the main diagonal). See also `tril(A,k)`.
- `triu(A)`: Extracts the upper triangular part of `A` (all entries above the main diagonal). See also `triu(A,k)`.
- `reshape(A,m,n)`: Returns an $m \times n$ matrix whose elements are taken columnwise from `A`. The original matrix must have $m * n$ elements. If you denote one of the indices m or n by square brackets [], the proper value will be computed automatically.

- `repmat(A,m,n)`: Creates an $m \times n$ tiling of copies of `A`. If A is of size $p \times q$, then the output will have size $mp \times nq$.
- `blkdiag(A1,A2,...)`: Creates a block diagonal matrix with blocks `A1`, `A2`, etc. on the diagonal.
- `norm(A)`: Returns the 2-norm of `A`, equal to the largest singular value of `A`. Use `norm(A,1)`, `norm(A,Inf)`, `norm(A,'fro')` for the 1-norm, infinity-norm, and Frobenius norm.
- `normalize(A,'norm')`: Scales the columns of `A` to have unit 2-norm.

3.9 Exercises

1. Why is $\|\mathbf{x}\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$ not a norm for $0 < p < 1$?

Part Two

4. Probability

4.1 Axioms

We start with some basic definitions and notation.

Definition 4.1.1 A *sample space* Ω is the set of possible *outcomes* of an experiment or trial.

Definition 4.1.2 An *event* E is a subset of Ω, containing zero or more outcomes.

The complement of an event E will be denoted by E^c, and the symbol $\emptyset$ will denote the empty set. Do not confuse the empty set with the number zero!

The union and intersection of two events are denoted $E \cup F$ and $E \cap F$, respectively. The difference of two sets is denoted $E \setminus F$, and is equal to $E \cap F^c$.

Definition 4.1.3 A *probability measure* $\mathbb{P}$ is a function that gives a probability to each event, having the following properties:

1. $\mathbb{P}(E) \geq 0$ for any event E. Events cannot happen with negative probability.
2. $\mathbb{P}(\Omega) = 1$. The probability that something happens is 1.
3. If $E_1, E_2, E_3, \ldots$ are mutually exclusive events ($E_i \cap E_j = \emptyset$ for all $i \neq j$), then

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} E_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(E_i).$$

Probabilities add over disjoint events.

■ **Example 4.1** Flip two distinguishable coins and record the results. Let Ω be the set of all outcomes, and let E be the event in which exactly one of the coins lands on heads. Then

$$\Omega = \{TT, TH, HT, HH\},$$
$$E = \{TH, HT\}.$$

Assuming the coins are both fair, $\mathbb{P}(E) = 1/2$. ■

R What exactly does it mean to say that a coin "has a 50 percent chance of landing on heads"? The definitions above sidestep this question by defining events and probabilities entirely in terms of sets and functions.

4.1.1 Consequences

From the axioms of a probability measure given in Definition 4.1.3, we can derive a few basic results.

Theorem 4.1.1 If $E \subseteq F$, then $\mathbb{P}(E) \leq \mathbb{P}(F)$. In particular, $\mathbb{P}(E) \leq 1$ for all events E.

Proof. The event F is the disjoint union of E and $F \setminus E$, and by Axiom 1, $\mathbb{P}(F \setminus E) \geq 0$. Thus by Axiom 3,

$$\mathbb{P}(F) = \mathbb{P}(E) + \mathbb{P}(F \setminus E) \geq \mathbb{P}(E).$$

The second statement follows by letting $F = \Omega$. ■

Theorem 4.1.2 For any event E, $\mathbb{P}(E^c) = 1 - \mathbb{P}(E)$. In particular, $\mathbb{P}(\emptyset) = 0$.

Theorem 4.1.3 — Inclusion-Exclusion. For any events E and F,

$$\mathbb{P}(E \cup F) = \mathbb{P}(E) + \mathbb{P}(F) - \mathbb{P}(E \cap F).$$

Finally, a handy theorem that applies in a wide range of scenarios such as flipping fair coins, rolling fair dice, or drawing cards from a deck.

Theorem 4.1.4 If the sample space Ω is finite and all outcomes in Ω are equally likely, then the probability of an event E is given by the formula

$$\mathbb{P}(E) = \frac{|E|}{|\Omega|},$$

where $|\cdot|$ denotes the number of elements in the set.

Unless explicitly stated otherwise, a "coin" is a fair coin, a "die" is a fair six-sided die, and a "card" is drawn from a standard 52-card deck, each card with equal probability.

■ **Example 4.2** Roll two dice. What is the probability of getting a sum of 4?

Let Ω denote all possible pairs of outcomes, so that $|\Omega| = 6 \times 6 = 36$. All of these outcomes are equally likely, and the ones that sum to 4 are $(1,3)$, $(2,2)$, and $(3,1)$, so by Theorem 4.1.4 the probability of getting a sum of 4 is $3/36 = 1/12$.

One could alternately try to approach the problem by treating $(1,3)$ and $(3,1)$ as the same outcome, in which case $|\Omega| = 21$ and two outcomes in Ω—$(1,3)$ and $(2,2)$—sum to 4. However, Theorem 4.1.4 does *not* apply since the outcomes $(1,3)$ and $(2,2)$ are not equally likely, and so $2/21$ is not a correct answer. ■

4.2 Discrete Random Variables

Definition 4.2.1 A *random variable* is a function $X : \Omega \to \mathbb{R}$. If $\omega \in \Omega$ is a possible outcome of a trial, then $X(\omega)$ is a value that we associate with that outcome.

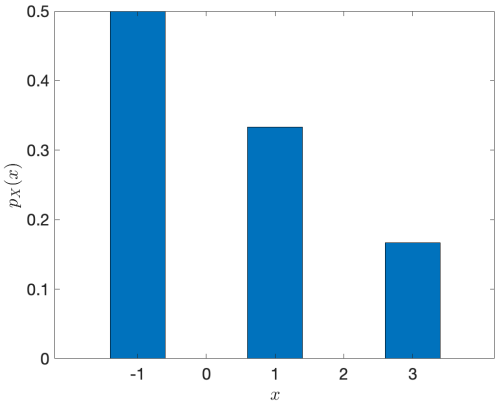


Figure 4.1: Plot of PMF from example 4.4

In general, we will use the capital Roman letters X, Y, Z, W for random variables. We will also use the notation $\mathbb{P}(X = a)$ as shorthand for $\mathbb{P}(E)$, where the event E is defined as $E = \{\omega \in \Omega : X(\omega) = a\}$. Cases such as $\mathbb{P}(X \leq a)$ or $\mathbb{P}(X \in [a,b])$ are defined similarly.

■ **Example 4.3** Flip 2 coins, and let X denote the number of heads. Then $X(HH) = 2$, $X(HT) = X(TH) = 1$, and $X(TT) = 0$. If the coin is fair, then $\mathbb{P}(X = 1) = 1/2$ and $\mathbb{P}(X \leq 1) = 3/4$. ■

Definition 4.2.2 A random variable X is *discrete* if it takes on a finite or countable number of distinct values.

Definition 4.2.3 The *probability mass function* (PMF) of a discrete random variable X is a function $p_X : \mathbb{R} \to [0,1]$ defined as

$$p_X(a) = \mathbb{P}(X = a), \quad \forall a \in \mathbb{R}.$$

In other words, $p_X(a)$ is the probability that X takes on the value a.

■ **Example 4.4** Roll a die. If you roll a 1, 2, or 3, you lose a dollar. If you roll a 4 or 5, you win a dollar. If you roll a 6, you win three dollars. If Y is the random variable representing your winnings, then Y takes on values in the set $\{-1,1,3\}$, but not with equal probability. Its PMF is

$$P_Y(x) = \begin{cases} 1/2 & x = -1, \\ 1/3 & x = 1, \\ 1/6 & x = 3, \end{cases}$$

and zero at all other values of x. A plot of the PMF is shown in Figure 4.1. ■

4.2.1 Common Discrete Distributions

A few discrete probability distributions are very common. Here we cover a few of them.

Definition 4.2.4 The *discrete uniform distribution* gives equal weight to each integer between two given integers a and b. If X is a discrete uniform random variable on this range, then we write $X \sim \text{Unif}\{a,b\}$ and have

$$p_X(k) = \frac{1}{b-a+1}, \quad a \leq k \leq b.$$

It is worth noting that we cannot define the discrete uniform distribution over an infinite range. We can, however, define a distribution over a countably infinite set so that the PMF is positive for each number in the set. The geometric distribution in Definition 4.2.9 is one example of such a distribution.

Definition 4.2.5 The *Bernoulli distribution* models an experiment with only two outcomes (typically encoded as 0 and 1). If X is a Bernoulli random variable succeeding with probability p, then we write $X \sim \text{Ber}(p)$ and have

$$p_X(x) = \begin{cases} p & x = 1, \\ 1-p & x = 0. \end{cases}$$

When the sample space Ω is defined explicitly, Bernoulli variables can be viewed in terms of indicator functions.

Definition 4.2.6 For an subset $E \subseteq \Omega$, the *indicator function* $\mathbf{1}_E$ is defined as

$$\mathbf{1}_E(\omega) = \begin{cases} 1 & \omega \in E, \\ 0 & \omega \notin E. \end{cases}$$

That is, when Ω is a sample space and E an event, the indicator function $\mathbf{1}_E$ is a Bernoulli random variable. Conversely, for any Bernoulli random variable X, we can define an event $E = \{\omega : X(\omega) = 1\}$, the set of all outcomes for which X takes on the value 1.

Definition 4.2.7 The *binomial distribution* represents the number of successes we get in a series of independent $0/1$ trials. If X is a binomial random variable with n trials and success probability p, then we write $X \sim \text{Bin}(n, p)$ and have

$$p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad 0 \le k \le n.$$

Figure 4.2 shows the PDF of a binomial distribution over 10 trials with success probability $p = 0.65$.

The notation $\binom{n}{k}$, called the binomial coefficient and read as "n choose k", represents the number of ways to choose k elements from a set of n distinct elements, where the order in which the elements are chosen does not matter. It can be defined in the following manner.

Definition 4.2.8 For an integer $n \ge 1$, the *factorial* is defined as

$$n! = n(n-1)\cdots 1, \quad 0! = 1,$$

and read as "n factorial". For an integer $0 \le k \le n$, the *binomial coefficient* is defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!}.$$

The binomial coefficient takes its name from its appearance as a coefficient in the binomial formula.

Theorem 4.2.1 — Binomial formula. For any two real numbers $x, y \in \mathbb{R}$ and any integer $n \ge 0$,

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}.$$

■ **Example 4.5** A biased coin lands on heads 80 percent of the time. If the coin is flipped 10 times, what is the probability of getting exactly 7 or 8 heads?

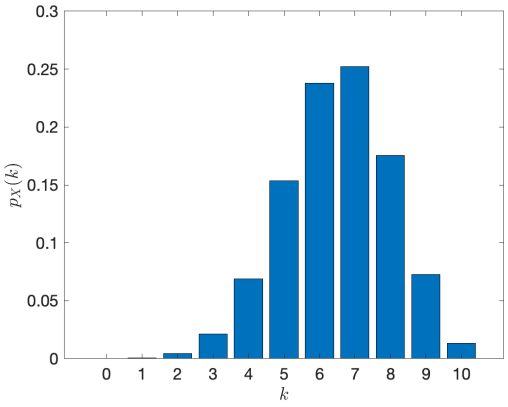


Figure 4.2: Binomial distribution over 10 trials with success probability $p = 0.65$.

Getting 7 heads and getting 8 heads are mutually exclusive events, so we can find the answer of getting one or the other by adding the individual probabilities. Since the number of heads X is a binomial random variable, we find that

$$\mathbb{P}(X = 7 \text{ or } X = 8) = \mathbb{P}(X = 7) + \mathbb{P}(X = 8) = \sum_{k=7}^{8} \binom{10}{k} (0.8)^k (0.2)^{10-k} \approx 0.503.$$

In Matlab, the answer can be found as `p = binopdf(7,10,0.8) + binopdf(8,10,0.8).` ■

■ **Example 4.6** A biased coin lands on heads 55 percent of the time. Which is more likely: getting 60 heads out of 100 flips, or 50 heads out of 100 flips?

Let $X \sim \text{Bin}(100, 0.55)$. Using the formula for the PMF, we find that

$$\mathbb{P}(X = 50) = \binom{100}{50} p^{50}(1-p)^{50} \approx 0.0482,$$
$$\mathbb{P}(X = 60) = \binom{100}{60} p^{60}(1-p)^{40} \approx 0.0488.$$

Thus it is more likely that we will get 60 heads. ■

The final distribution shows that it is possible to have a discrete distribution that can take on infinitely many possible values.

Definition 4.2.9 The *geometric distribution* represents the number of failures we encounter before getting a single success when running independent $0/1$ trials. If X is a geometric random variable with success probability p, then we write $X \sim \text{Geo}(p)$ and have

$$p_X(k) = (1-p)^k p, \quad k \geq 0.$$

■ **Example 4.7** If we roll a die until we get a 6, what is the probability that we will have 5, 6, or 7 failures first?

Using the random variable $X \sim \text{Geo}(1/6)$, we get that

$$\mathbb{P}(5 \leq X \leq 7) = \sum_{k=5}^{7} (5/6)^k (1/6) \approx 0.169.$$

In Matlab, we can use the command `p = sum(geopdf(5:7,1/6))`. ■

4.3 Continuous Random Variables

We begin by defining the cumulative distribution function of a random variable.

Definition 4.3.1 For a real-valued random variable X, the *cumulative distribution function* (CDF) of X is a function $F_X : \mathbb{R} \to [0,1]$ such that for any $x \in \mathbb{R}$,

$$F_X(x) = \mathbb{P}(X \leq x).$$

Every real-valued random variable has a CDF, but it is possible for two different random variables to have the same CDF.

■ **Example 4.8** Flip a coin. Let X be the number of heads, and let $Y = 1 - X$ be the number of tails. Then X and Y have the same $\text{Ber}(1/2)$ distribution, with CDF

$$F_X(x) = F_Y(x) = \begin{cases} 0 & x < 0, \\ 1/2 & 0 \leq x < 1, \\ 1 & x \geq 1. \end{cases}$$

But $X(\omega) \neq Y(\omega)$ for all outcomes $\omega \in \Omega = \{T, H\}$. ■

Fact 4.3.1 The CDF F_X of a random variable X has the following properties:

1. (Limiting values) $\lim_{x \to -\infty} F_X(x) = 0$ and $\lim_{x \to \infty} F_X(x) = 1$.
2. (Monotonic) If $a \leq b$ then $F_X(a) \leq F_X(b)$.
3. (Right-continuous) For any $a \in \mathbb{R}$, $F(a) = \lim_{x \to a^+} F(x)$.

Example 4.8 shows that the CDF need not be left-continuous.

Fact 4.3.2 For any random variable X, the CDF F_X satisfies

$$\mathbb{P}(a < X \leq b) = F_X(b) - F_X(a).$$

Definition 4.3.2 A *continuous random variable* is a random variable whose CDF is everywhere continuous.

If the CDF of a random variable is absolutely continuous, then the random variable also has what is called a probability density function.

Definition 4.3.3 A random variable X has a *probability density function* (PDF) $f_X : \mathbb{R} \to [0, \infty)$ if

$$\mathbb{P}(a \leq X \leq b) = \int_{x=a}^{b} f_X(x)\, dx.$$

Fact 4.3.3 The PDF and CDF are related in the following manner: for all $a \in \mathbb{R}$,

$$F_X(a) = \int_{t=-\infty}^{a} f_X(t)\, dt.$$

For all points x at which F_X is differentiable,

$$f_X(x) = \frac{d}{dx} F_X(x).$$

Intuitively, we can think of $f_X(x)\,dx$ as the probability that X takes on a value in the infinitesimal interval $[x, x+dx]$. The ratio $f_X(a) : f_X(b)$ can be interpreted as the relative likelihood of X taking on the value a versus b, but bear in mind that the probability of X taking on any single exact value is zero.

Fact 4.3.4 If X is a continuous random variable, then for any $a \in \mathbb{R}$,

$$\mathbb{P}(X = a) = 0.$$

Roughly, if we sample a random variable many times, then the histogram of sampled values should have a shape resembling the PDF. For those interested, the Glivenko-Cantelli theorem makes this notion more precise. Fact 4.3.3 also implies the following property of the PDF:

Fact 4.3.5 The PDF f_X of a random variable X satisfies

$$\int_{t=-\infty}^{\infty} f_X(t)\,dt = 1.$$

In other words, X will take on a value in the range $(-\infty, \infty)$ with probability 1.

■ **Example 4.9** Throw a dart at a circular dartboard so that it lands according to the uniform distribution: that is, if Ω is the set of points in the circle and E is a designated area, then the probability of landing in E is $\text{vol}(E)/\text{vol}(\Omega)$. Let X be a random variable representing the distance of the dart from the center of the circle. What are the PDF and CDF of X?

It's simplest to start with the CDF. Let r be the radius of the dartboard, so that the area is πr^2. Then for any $0 \le a \le r$,

$$F_X(a) = \mathbb{P}(X \le a) = \frac{\pi a^2}{\pi r^2} = \frac{a^2}{r^2},$$

since the condition $X \le a$ defines a circle of radius a. Then $F_X(a) = 0$ for $a \le 0$ and $F_X(a) = 1$ for $a \ge r$. Taking the derivative, we find that the PDF is

$$f_X(a) = \begin{cases} 0 & a \le 0, \\ \frac{2a}{r^2} & 0 < a \le r, \\ 0 & a > r. \end{cases}$$

We note that it does not matter exactly how f_X is defined at its points of discontinuity. In any case, the implication is that the dart is more likely to land far from the center than close to it. ■

■ **Example 4.10** Suppose X is a random variable with the PDF

$$f_X(x) = \begin{cases} \alpha(1-x^2) & -1 \le x \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

What value of α makes f_X a valid probability distribution? What is the CDF? What is $\mathbb{P}(X \ge 3/4)$?

The PDF must integrate to 1, so

$$1 = \int_{x=-1}^{1} \alpha(1-x^2)\,dx = \alpha\left[1 - \frac{x^3}{3}\right]_{-1}^{1} = \frac{4}{3}\alpha,$$

so $\alpha = 3/4$. Second,

$$F_X(a) = \int_{-\infty}^{a} f_X(x)\,dx = \begin{cases} 0 & x \le -1, \\ \frac{1}{2} + \frac{3}{4}a - \frac{1}{4}a^3 & -1 < x \le 1, \\ 1 & x > 1. \end{cases}$$

As a sanity check, we can verify that the middle formula gives $F(-1) = 0$, $F(0) = 1/2$ (the PDF is symmetric!), and $F(1) = 1$. Finally,

$$\mathbb{P}(X \geq 3/4) = 1 - \mathbb{P}(X \leq 3/4) = 1 - F(3/4) = 11/256 \approx 0.043.$$

■

4.3.1 Common Continuous Distributions

Here we cover a few of the most common continuous distributions.

Definition 4.3.4 The *uniform distribution* Unif$[a,b]$ has PDF

$$f(x) = \begin{cases} \frac{1}{b-a} & x \in [a,b], \\ 0 & \text{otherwise} \end{cases}$$

and CDF

$$F(x) = \begin{cases} 0 & x \leq a, \\ \frac{x-a}{b-a} & a < x \leq b, \\ 1 & x > b. \end{cases}$$

Note that if $b-a > 1$, $f(x)$ can take on values greater than 1. Unlike probabilities (and thus the CDF), probability *densities* do not have to lie in the interval $[0,1]$.

■ **Example 4.11** Spin a spinner. If any stopping point is assumed to be equally likely, then the angle the spinner makes with respect to a fixed reference point can be modeled as a Unif$[0, 2\pi]$ random variable. ■

Definition 4.3.5 A *normal* or *Gaussian* distribution Normal(μ, σ^2) has the PDF

$$\phi(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}.$$

The *standard normal* or *standard Gaussian* distribution uses $\mu = 0$, $\sigma^2 = 1$.

Given parameters μ and σ^2 this distribution will be the famous bell curve, centered at μ. The parameter σ controls how tightly concentrated the distribution is around μ—more to come in the following chapters. The CDF of the normal distribution does not have a nice closed-form expression, but software packages have been designed to compute it accurately.

Notation 4.1. *The CDF of the standard normal distribution will be denoted by* Φ.

The fact that the PDF of the normal distribution integrates to 1 (as do all PDFs) implies the following fact. We leave the proof for another course.

Fact 4.3.6

$$\int_{-\infty}^{\infty} e^{-x^2/2}\, dx = \sqrt{2\pi}.$$

■ **Example 4.12** Let $X \sim$ Unif$[0,1]$ and let $Y = X^2$. What is the PDF of Y?

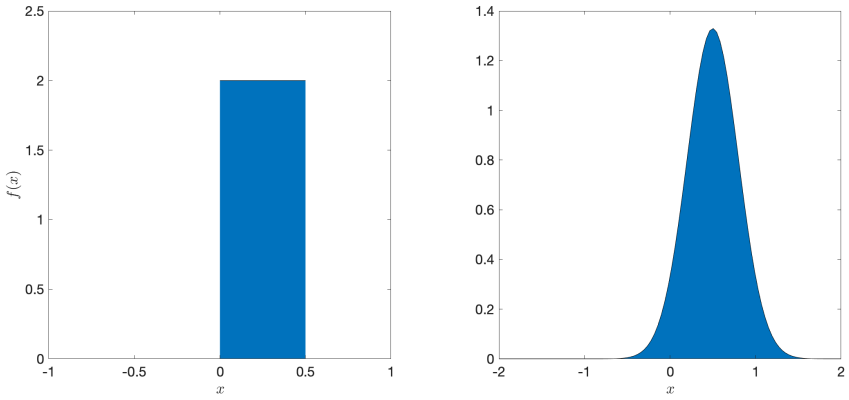


Figure 4.3: Left: Uniform distribution on $[0, \frac{1}{2}]$. Right: Gaussian with $\mu = 0.5$ and $\sigma = 0.3$.

To answer this, we can consider the CDFs of the two variables. For any $a \in [0,1]$,

$$\begin{aligned} F_Y(a) &= \mathbb{P}(Y \leq a) \\ &= \mathbb{P}(X^2 \leq a) \\ &= \mathbb{P}(X \leq \sqrt{a}) \\ &= \sqrt{a}. \end{aligned}$$

Taking the derivative, we find that the PDF of Y is

$$f_Y(x) = \begin{cases} 0 & x \leq 0, \\ \frac{1}{2\sqrt{x}} & 0 < x \leq 1, \\ 0 & x > 1. \end{cases}$$

Interestingly, this example shows that the PDF does not even need to be bounded! ■

4.4 Independence

The definition of the binomial distribution (Definition 4.2.7) stipulated that the trials must be *independent*. Informally, the outcome of any one trial should not tell us anything about the outcome of the others. Here we make that notion more precise.

Definition 4.4.1 Two events E_1 and E_2 are *independent* if $\mathbb{P}(E_1 \cap E_2) = \mathbb{P}(E_1)\mathbb{P}(E_2)$.

■ **Example 4.13** Draw a card from a standard deck. If E_1 is the event "draw a club" and E_2 is the event "draw a seven", then $\mathbb{P}(E_1 \cap E_2) = 1/52$ ("draw the seven of clubs") and $\mathbb{P}(E_1)\mathbb{P}(E_2) = (1/4)(1/13) = 1/52$. Thus the events are independent. ■

■ **Example 4.14** Technically, for any event E, the events E and Ω are independent. Similarly, E and $\emptyset$ are independent. ■

When more than two events are involved, things get slightly more complicated.

Definition 4.4.2 A finite set of events $\{E_i\}_{i=1}^n$ is *pairwise independent* if each pair of events is independent.

Definition 4.4.3 A finite set of events $\{E_i\}_{i=1}^n$ is *mutually independent* if for every k-element subset $\{j_1,\dots,j_k\} \subset \{1,\dots,n\}$,

$$\mathbb{P}\left(\bigcap_{i=1}^{k} E_{j_i}\right) = \prod_{i=1}^{k} \mathbb{P}(E_{j_i}).$$

Mutual independence is a strictly stronger condition than pairwise independence.

■ **Example 4.15** Flip two coins. Let E_1 be the event that the first coin lands on heads, let E_2 be the event that the second coin lands on heads, and let E_3 be the event that exactly one of the two coins lands on heads. Then the events are pairwise independent but not mutually independent. ■

■ **Example 4.16** If we draw several cards from a deck one at a time but do not replace the cards, then the successive outcomes are *not* independent. For example, the probability of drawing the ace of spaces as the first card is 1/52. The probability of drawing it as the second card is 1/51 if it was not chosen on the first draw, or zero if it was. ■

R The *gambler's fallacy* is the fallacy that if an event has occurred more frequently than normal in the past, then it is less likely to occur in the future. If a coin lands on heads several times in a row, it does not make the next flip any more likely to come up tails. It might be rational to reason that the coin might be biased and therefore more likely to land on heads again, but the coin itself does not carry any "memory" of past outcomes.

4.4.1 Independent random variables

Definition 4.4.4 Two random variables X and Y are *independent* if for all $x, y \in \mathbb{R}$,

$$F_{X,Y}(x,y) = F_X(x)F_Y(y).$$

Here $F_{X,Y}(x,y) = \mathbb{P}(X \le x \text{ and } Y \le y)$ is the *joint CDF* of X and Y.

Intuitively, learning the value of X tells you nothing at all about the value of Y.

Definition 4.4.5 A finite set of random variables $\{X_i\}_{i=1}^n$ is *pairwise independent* if each pair of random variables is independent. The set is *mutually independent* if for all $x_1,\dots,x_n \in \mathbb{R}$,

$$F_{X_1,\dots,X_n}(x_1,\dots,x_n) = F_{X_1}(x_1)\cdots F_{X_n}(x_n).$$

Fact 4.4.1 When X and Y are discrete random variables, the independence condition is equivalent to

$$\mathbb{P}(X = x \text{ and } Y = y) = \mathbb{P}(X = x)\mathbb{P}(Y = y).$$

When X and Y are continuous random variables with a *joint PDF* $f_{X,Y}$, the independence condition is equivalent to

$$f_{X,Y}(x,y) = f_X(x)f_Y(y).$$

Definition 4.4.6 The random variables $X_1,\dots,X_n$ are *independent and identically distributed* (i.i.d.) if they have the same probability distribution and are mutually independent. If they come from a common distribution D, we may use the notation $X_1,\dots,X_n \overset{i.i.d.}{\sim} D$.

■ **Example 4.17** A sequence of coin flips, dice rolls, or spins of a roulette wheel are generally

assumed to be i.i.d. random variables. This holds regardless of whether the coins/dice/wheel are fair or loaded. ■

■ **Example 4.18** A binomial random variable $X \sim \text{Bin}(n,p)$ can be written as the sum of i.i.d. Bernoulli random variables

$$X = \sum_{i=1}^{n} X_i, \quad X_1, \ldots, X_n \overset{i.i.d.}{\sim} \text{Ber}(p).$$

■

4.5 Joint probability density function

For completeness, we define the joint PMF and joint PDF. Starting with discrete random variables:

Definition 4.5.1 The *joint probability mass function* of two discrete random variables X and Y is a function $p_{X,Y} : \mathbb{R}^2 \to [0,1]$ satisfying

$$p_{X,Y}(x,y) = \mathbb{P}(X = x \text{ and } Y = y), \quad (x,y) \in \mathbb{R}^2.$$

The joint PMF is defined for any pair of discrete random variables.

Fact 4.5.1 If the discrete random varaibles X and Y have the joint PMF $p_{X,Y}$, then the individual PMFs are given by

$$p_X(x) = \sum_y p_{X,Y}(x,y),$$
$$p_Y(y) = \sum_x p_{X,Y}(x,y).$$

By Fact 4.4.1, X and Y are independent if and only if $p_{X,Y}(x,y) = p_X(x)p_Y(y)$.

Joint distributions can be defined for continuous random variables similarly, although not every pair of continuous random variables has a joint probability density function.

Definition 4.5.2 The continuous random variables X and Y have a *joint probability density function* $f_{X,Y} : \mathbb{R}^2 \to [0,\infty)$ if

$$\mathbb{P}(a < X \le b, c < Y \le d) = \int_{x=a}^{b} \int_{y=c}^{d} f_{X,Y}(x,y)\, dy\, dx.$$

The joint PDF has several properties that make it a natural extension of the PDF for a single variable.

Fact 4.5.2 The joint PDF $f_{X,Y}$ of a pair of random variables X and Y satisfies

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{X,Y}(x,y)\, dy\, dx$$

Fact 4.5.3 The joint PDF and joint CDF are related in the following manner:

$$F_{X,Y}(x,y) = \int_{s=-\infty}^{x} \int_{t=-\infty}^{y} f_{X,Y}(s,t)\, dt\, ds.$$

For all points $(x,y) \in \mathbb{R}^2$ at which $F_{X,Y}$ is differentiable,

$$f_{X,Y}(x,y) = \frac{\partial^2 F_{X,Y}(x,y)}{\partial x \partial y}.$$

Fact 4.5.4 If the random variables X and Y have the joint PDF $f_{X,Y}$, then the marginal distributions are given by

$$f_X(a) = \int_{-\infty}^{\infty} f_{X,Y}(a,y)\,dy,$$
$$f_Y(b) = \int_{-\infty}^{\infty} f_{X,Y}(x,b)\,dx.$$

The joint PDF can be defined for higher dimensions in a similar manner.

■ **Example 4.19** Let $X, Y \overset{i.i.d.}{\sim} \text{Unif}[0,1]$. What is $\mathbb{P}(Y \leq X^2)$?

This problem can be solved by setting up a 2-dimensional integral and integrating the joint PDF $f_{X,Y}$ over the domain $\{(x,y) \in [0,1]^2 \,:\, y \leq x^2\}$.

$$\begin{aligned}
\mathbb{P}(Y \leq X^2) &= \int_{x=0}^{1} \int_{y=0}^{x^2} f_{X,Y}(x,y)\,dy\,dx \\
&= \int_{x=0}^{1} \int_{y=0}^{x^2} f_X(x) f_Y(y)\,dy\,dx \\
&= \int_{x=0}^{1} \int_{y=0}^{x^2} 1\,dy\,dx \\
&= \int_{x=0}^{1} x^2\,dx \\
&= \frac{1}{3}.
\end{aligned}$$

This answer relies on the assumption that X and Y are independent. If, for example, we had $Y = X$, then the probability would be zero. If instead we had $Y = 1 - X$, the probability would be $(3-\sqrt{5})/2 \approx 0.382$. ■

4.6 Bertrand's Paradox

A fun little problem to illustrate the importance of defining probability distributions:

> Draw a circle of radius 1 inside a circle of radius 2, both having the same center. What is the probability that a random chord of the larger circle intersects the smaller circle?

There are a few different ways we could consider generating a random chord. First, we could pick a point uniformly at random in the interior of the larger circle. There is then a unique chord having this point as its midpoint. If we follow this strategy, then the probability is 1/4 because the small circle has 1/4 the area of the large circle.

A second approach is to pick two points at random on the boundary of the larger circle, then let the chord be the line segment connecting them. To find the probability with this approach, fix one of the points and draw the two chords that intersect that point and are tangent to the inner circle. These two chords sweep out a 120-degree arc of the circle, so there is a $120/360 = 1/3$ chance that the second point falls on this arc. Thus the probability is 1/3.

A third approach is to pick a random diameter of the larger circle, then let the midpoint of the chord be a point chosen randomly along the diameter. The outer circle has diameter 4 and the inner

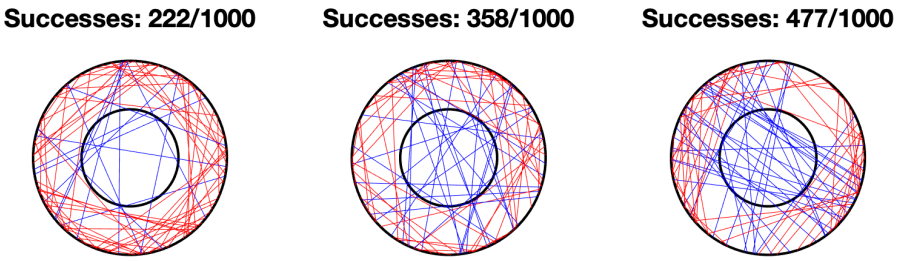


Figure 4.4: Left: pick midpoint uniformly at random. Center: pick two endpoints. Right: pick diameter, then midpoint randomly along the diameter.

circle has diameter 2, so the probability that a random point on the outer diameter will fall inside the inner circle is 1/2. Figure 4.4 reports the results of 1000 trials using each method, with the first 75 trials of each method being plotted.

So why did we get three different answers, and which one is correct? The answer is that there isn't any single correct way to pick a "random" chord. There are infinitely many different ways, all associated with different probability distributions. You could potentially argue that one distribution is more "natural" than the others, but the problem doesn't specify which distribution should be used.

4.7 Matlab Commands

Discrete Random Variables

- `randi(K)`: Generates a random integer from the Unif$\{1,K\}$ distribution. Use `randi(J,K)` to sample from Unif$\{J,K\}$. See also `unidrnd` for similar functionality.
- `unidpdf, unidcdf, unidinv`: PMF, CDF, and inverse CDF for discrete uniform distribution. Strictly speaking, the CDF is not invertible so `unidinv(p)` returns the least integer a such that $\mathbb{P}(X \leq a) \geq p$.
- `binornd(n,p)`: Random number from a Bin(n,p) distribution.
- `binopdf,binocdf,binoinv`: PMF, CDF, and inverse CDF of binomial distribution. Same caveat with CDF not being invertible.
- `geornd(p)`: Random number from a Geo(p) distribution.
- `geopdf,geocdf,geoinv`: PMF, CDF, and inverse CDF of geometric distribution. Same caveat with CDF not being invertible.

Continuous Random Variables

- `rand`: Generates a random Unif$[0,1]$ number.
- `unifrnd(a,b)`: Generates a random Unif(a,b) number.
- `unifpdf, unifcdf, unifinv`: PDF, CDF, inverse CDF of uniform distribution.
- `randn`: Generates a random Normal$(0,1)$ number.
- `normrnd(mu,sigma)`: Generates a random Normal(μ,σ^2) number.
- `normpdf,normcdf,norminv`: PDF, CDF, inverse CDF of normal distribution.

Miscellany

- `rng(S)`: Seeds the random number generator using the seed `S`. Useful for ensuring that your code is reproducible.

- `randsample`: Random sampling from a population, with or without replacement.
- `randperm(N,K)`: Returns a row vector containing K unique integers from $1:N$. Use `randperm(N)` to get a random permutation of the indices $1:N$.
- `factorial(n)`: Returns $n!$
- `nchoosek(n,k)`: Returns $\binom{n}{k}$. Using `nchoosek(v,k)` where v is a vector of length n returns an array containing all $\binom{n}{k}$ subsets of v with size k.
- `random`: Generate random numbers for a wide variety of distributions. See documentation for options.
- `pdf,cdf,icdf`: PDF/PMF, CDF, and inverse CDF for a wide variety of distributions.

4.8 Exercises

1. Prove Theorem 4.1.2.
2. Prove Theorem 4.1.3.
3. Let $X \sim \text{Unif}[0,1]$, and let $Y = aX + b$. By examining the CDF of Y, show that $Y \sim \text{Unif}[b, a+b]$.
4. Let $X \sim \text{Normal}(0,1)$, and let $Y = aX + b$. By examining the CDF of Y, show that $Y \sim \text{Normal}(b, a^2)$.
5. Suppose X is a random variable with the PDF

   $$f_X(x) = \max\{0, 3 - \alpha|x|\}.$$

   What value of α makes this a valid PDF? What is the resulting CDF? What is $\mathbb{P}(X \leq 1/10)$?
6. You are playing a video game in which an attack is said to hit 25% of the time. Assuming the number of failures between hits is modeled by a geometric distribution, what is the probability that you will fail at least 8 times in a row in a given series of attacks?
7. (Ipsen) A plane is able to land safely if at least half of the engines function properly. Let $0 < p < 1$ be the probability that a given engine functions properly. If the number of functioning engines on an n-engine plane is (unrealistically) modeled as $\text{Bin}(n, p)$, for what values of p is a 2-engine plane safer than a 4-engine plane?
8. Suppose the lifespan of a lightbulb is modeled by the distribution $f_X(t) = \alpha e^{-\lambda t}$ for $t > 0$, and 0 for $t \leq 0$, where λ is a parameter related to the quality of the bulb. (a) What value of α makes the PDF valid? (b) What is the CDF? (c) What is the median lifespan of the bulb? This is the value m such that $\mathbb{P}(X \geq m) = 0.5$. (d) How is λ related to the quality of the bulb?
9. Why is it not possible to define the discrete uniform distribution over an infinite set of values? Use the axioms of probability to justify your reasoning.

5. Properties of Random Variables

Who's in the mood for a little wager?

■ **Game 1** Flip a coin. If it lands on heads, you win a nice shiny nickel. Tails, you get an automatic F for the semester.

What if the prize goes up to 10 dollars? 100? Ten thousand? A million? Is there any amount of money on offer that would entice you to take this bet?

■ **Game 2** Flip a coin 100 times. If it lands on heads at least 40 times, you win a nice shiny nickel. If not, you get an automatic F for the semester.

Seems a bit risky. What if the number of heads needed is lowered to 30? 20? 10? Just *1*?? When is the downside improbable enough that you are willing to take the risk?

Here are a few games that are a little bit tamer. How much would you be willing to pay in order to play each of these games?

■ **Game 3** Roll a die, and collect X dollars for rolling the number X.

■ **Game 4** Roll a die, and collect 21 dollars if you roll a 6.

■ **Game 5** Roll a die, and collect \$3.50 no matter what!

5.1 Expected Value

In order to talk about the tradeoffs involved in these games rigorously, we need to add a few more definitions to our vocabulary. In particular, we want *summary statistics* that are capable of summarizing the information about a probability distribution as simply as possible.

Definition 5.1.1 — Discrete case. The *expected value* of a discrete random variable X is defined as

$$\mathbb{E}[X] = \sum_x x p_X(x),$$

where the sum is taken over all possible outcomes of X. This definition is valid as long as the value of the sum does not depend on the order of summation—otherwise, the expected value is not well defined.

Fact 5.1.1 If the sample space Ω is countable with probability measure $\mathbb{P}$, we can alternately define the expected value as

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega)\mathbb{P}(\omega).$$

This again assumes that the sum is independent of the order of summation.

This second formula is not necessarily cleaner, but it more closely resembles the formula for continuous random variables.

Definition 5.1.2 — Continuous case. If X is a random variable with PDF f_X, the expected value of X is

$$\int_{-\infty}^{\infty} x f_X(x)\,dx,$$

assuming that the integral is finite over at least one of the domains $[0,\infty)$ and $(-\infty,0]$. Otherwise, the expected value is undefined.

The expected value $\mathbb{E}[X]$ effectively takes a weighted average over all possible outcomes for X, where the weighting is the probability of that outcome occurring. We will justify this concept in more detail later, but intuitively, it is something like the "fair" value for a game. It also corresponds closely with the center of mass in physics/engineering, representing the point around which the probability distribution is "balanced".

■ **Example 5.1** In Game 3, the expected value is

$$\mathbb{E}[X] = \sum_{i=1}^{6} i p_X(i) = \sum_{i=1}^{6} i/6 = 3.50.$$

Thus a "fair" value for the game would be \$3.50.

In Game 4, the expected value is

$$\mathbb{E}[X] = 21\left(\frac{1}{6}\right) + 0\left(\frac{5}{6}\right) = \frac{21}{6} = 3.50.$$

so the expected value is the same as for the previous game.

In Game 5, the expected value is

$$\mathbb{E}[X] = 3.5 \cdot 1 = 3.50.$$

It would be fair, but not very interesting, if you paid \$3.50 for a chance to play this game. ■

■ **Example 5.2** Consider the uniform random variable $X \sim \text{Unif}[0,1]$. Then

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\,dx = \int_{x=0}^{1} x\,dx = \frac{1}{2}x^2\Big|_0^1 = \frac{1}{2}.$$

This makes intuitive sense since the PDF is symmetric about the axis $x = 1/2$. ■

■ **Example 5.3** Consider a random variable X with PDF $f_X(x) = x^{-2}$ for $x \geq 1$. The expected value is

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\,dx = \int_{x=1}^{\infty} x^{-1}\,dx = \log(x)|_1^{\infty} = +\infty.$$

The expected value is infinite. ■

■ **Example 5.4** Consider a random variable X with the *Cauchy distribution*: having the PDF

$$f_X(x) = \frac{1}{\pi(1+x^2)}.$$

This distribution function is symmetric about the origin, so intuitively we might want to say that the expected value is zero. However,

$$\int_{-\infty}^{\infty} x f_X(x)\,dx = \int_{-\infty}^{\infty} \frac{x}{\pi(1+x^2)}\,dx = \frac{1}{2\pi}\log(1+x^2)\Big|_{-\infty}^{\infty},$$

which is not well defined. Therefore, the expected value of X is not well defined. ■

R Given the previous example, it is reasonable to wonder why expected value is defined the way it is. The reason has to do with concentration inequalities such as the Weak Law of Large Numbers (covered in a later chapter). We would like to be able to say something along the lines of "If you sample a variable repeatedly, then with high probability the sample average should be close to the expected value." This principle fails for the Cauchy distribution if we define its expected value to be zero.

5.1.1 Properties

The expected value of a constant random variable is that constant.

Fact 5.1.2 For any $a \in \mathbb{R}$, $\mathbb{E}[a] = a$.

One particularly important property of the expected value is that of *linearity*.

Theorem 5.1.3 For any random variables X and Y with well-defined expected values, and for any real numbers $a, b \in \mathbb{R}$,

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y].$$

Proof for discrete random variables. Using the alternative formula from Fact 5.1.1, we find that

$$\begin{aligned}\mathbb{E}[aX+bY] &= \sum_{\omega\in\Omega} (aX+bY)(\omega)\mathbb{P}(\omega) \\ &= a\sum_{\omega\in\Omega} X(\omega)\mathbb{P}(\omega) + b\sum_{\omega\in\Omega} Y(\omega)\mathbb{P}(\omega) \\ &= a\mathbb{E}[X] + b\mathbb{E}[Y].\end{aligned}$$

■

Another applies when dealing with independent random variables.

Theorem 5.1.4 If X and Y are independent random variables, then

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y].$$

The converse is not true. Just because $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ does not mean that X and Y are independent.

■ **Example 5.5** Roll two dice. Assuming the rolls are independent, what is the expected value of the product?

Let X and Y be the numbers rolled. Since the two are assumed to be independent, we conclude that

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = (3.5)^2 = 12.25.$$

■

Finally, we provide a result that is useful for when we want to find the expected value of $g(X)$ knowing only the distribution of X. It gets its name, supposedly, from the tendency to treat it as a definition rather than a result requiring proof.

Theorem 5.1.5 — Law of the unconscious statistician. If X is a discrete random variable with PMF p_X, then

$$\mathbb{E}[g(X)] = \sum_x g(x)p_X(x),$$

where the sum is over all outcomes x of X. If X is continuous with PDF f_X, then

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)f_X(x)\,dx.$$

■ **Example 5.6** Consider the Bernoulli random variable $X \sim \text{Ber}(p)$. Then

$$\mathbb{E}[X] = 1(p) + 0(1-p) = p.$$

■

■ **Example 5.7** Consider the Binomial random variable $X \sim \text{Bin}(n,p)$. Then as mentioned in Example 4.18, we can rewrite X as a sum of Bernoulli variables:

$$X = \sum_{i=1}^{n} X_i, \quad X_i \sim \text{Ber}(p).$$

That is, we are running n trials, and incrementing a counter by 1 for each trial that succeeds. Then by the linearity of expectation,

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} p = np.$$

So if we flip n coins, each with a p chance of landing on heads, we expect to get np heads. ■

■ **Example 5.8** Consider the standard Gaussian variable $X \sim \text{Normal}(0,1)$. What are $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$?

The expected value is given by the integral

$$\mathbb{E}[X] = \frac{1}{\sqrt{2\pi}} \int_{x=-\infty}^{\infty} xe^{-x^2/2}\,dx = -e^{-x^2/2}\Big|_{-\infty}^{\infty} = 0.$$

Note also that the normal distribution is symmetric about zero, so the expected value will be zero as long as the integral converges on $[0,\infty)$ and $(-\infty,0]$.

As for $\mathbb{E}[X^2]$, we get using Theorem 5.1.5 that

$$\begin{aligned}\mathbb{E}[X^2] &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} x^2 e^{-x^2/2}\,dx \\ &= \frac{1}{\sqrt{2\pi}}\left(-xe^{-x^2/2}\Big|_{-\infty}^{\infty} + \int_{-\infty}^{\infty} e^{x^2/2}\,dx\right) \\ &= \frac{1}{\sqrt{2\pi}}\left(0+\sqrt{2\pi}\right) \\ &= 1.\end{aligned}$$

The equality between the first and second lines uses integration by parts with $u = x$ and $dv = xe^{-x^2/2}\,dx$, and the step after that uses Fact 4.3.6 to evaluate the integral. ∎

5.2 Variance

The three games above all have the same expected value, but are quite different in character from one another. Clearly, just knowing the expected value doesn't tell us everything we might want to know about the distribution of a random variable. A second measurement might tell us how *concentrated* the probability distribution is around its expected value, which gives us some information about how likely we are to observe outliers. We want a number that answers the question

> How far, *on average*, is a random variable from its expected value?

There are many different ways we could measure this quantity, depending on how we decide to measure the "average". Perhaps the most conceptually straightforward variant is the mean absolute deviation.

Definition 5.2.1 The *mean absolute deviation* (MAD) of a random variable X is defined as

$$\mathrm{D}[X] = \mathbb{E}[|X - \mathbb{E}[X]|].$$

∎ **Example 5.9** Let $X \sim \mathrm{Unif}[0,1]$. Then $\mathbb{E}[X] = \int_0^1 x\,dx = 1/2$, and

$$\mathrm{D}[X] = \mathbb{E}[|X - 1/2|] = \int_0^1 |x - 1/2|\,dx = 1/4.$$

A uniform $[0,1]$ random variable is, on average, a distance of 1/4 from its expected value of 1/2. ∎

Another measurement is far more common due to its nicer mathematical properties.

Definition 5.2.2 The *variance* of a random variable X is defined as

$$\mathbb{V}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right],$$

and the *standard deviation* of X is defined as

$$\sigma(X) = \sqrt{\mathbb{V}(X)}.$$

The relation of the standard deviation to the MAD is analogous to the relation between the 2-norm and 1-norm of a vector. Both have the same units as the random variable. The variance is analogous to the squared 2-norm (as well as the moment of inertia in physics), and its units are the square of the random variable's units. For more details, see Fact 5.2.6.

∎ **Example 5.10** Let $X \sim \mathrm{Unif}[0,1]$, as before. Then

$$\mathbb{V}(X) = \mathbb{E}[(X - 1/2)^2] = \int_0^1 (x - 1/2)^2\,dx = 1/12.$$

Consequently, $\sigma(X) = \sqrt{1/12} \approx 0.289$. This is slightly larger than the MAD. ■

■ **Example 5.11** The expected value of Game 3 is 3.5, so the variance is

$$\begin{aligned}\mathbb{V}(X) &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\ &= \frac{1}{6}\sum_{i=1}^{6}(i-3.5)^2 \\ &= 35/12\end{aligned}$$

and the standard deviation is $\sigma(X) = \sqrt{35/12} \approx 1.71$. ■

■ **Example 5.12** The variance of Game 5 is zero, because the random variable is constant. ■

5.2.1 Properties

An alternate formula for the variance sometimes comes in handy.

Theorem 5.2.1 For any random variable X, $\mathbb{V}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

Proof. To simplify the notation, let $\mathbb{E}[X] = \mu$. Then by the linearity of the expected value (Theorem 5.1.3), we have

$$\begin{aligned}\mathbb{V}(X) &= \mathbb{E}\left[(X-\mu)^2\right] \\ &= \mathbb{E}\left[X^2 - 2\mu X + \mu^2\right] \\ &= \mathbb{E}[X^2] - 2\mu\mathbb{E}[X] + \mathbb{E}[\mu^2] \\ &= \mathbb{E}[X^2] - \mu^2.\end{aligned}$$

■

■ **Example 5.13** Let $X \sim \text{Ber}(p)$. Then using the formula of Theorem 5.2.1 along with Example 5.6, we find that

$$\mathbb{V}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = p - p^2 = p(1-p).$$

Note that since X takes on the values 0 and 1 only, it follows that $X^2 = X$. The standard deviation is then $\sigma(X) = \sqrt{p(1-p)}$. ■

Fact 5.2.2 For any random variable X and any $a, b \in \mathbb{R}$,

$$\mathbb{V}(aX+b) = a^2\mathbb{V}(X)$$

and

$$\sigma(aX+b) = |a|\sigma(X).$$

Note the absolute value in the second equation—the standard deviation and variance are always nonnegative!

■ **Example 5.14** In Game 4, the winnings can be modeled as $W = 21X$, where $X \sim \text{Ber}(1/6)$. Thus by Fact 5.2.2 and Example 5.13,

$$\mathbb{V}(W) = 21^2\mathbb{V}(X) = 21^2(1/6)(5/6) = 61.25$$

and $\sigma(W) = \sqrt{61.25} = 7\sqrt{5}/2 \approx 7.82$.

■

One key fact about the normal distribution in particular is that a linear transformation of a normally distributed random variable is still normally distributed.

Theorem 5.2.3 Let $X \sim \text{Normal}(\mu, \sigma^2)$. Then for any $a, b \in \mathbb{R}$ with $a \neq 0$, $aX + b \sim \text{Normal}(a\mu + b, a^2\sigma^2)$.

Fact 5.2.4 If $\mathbb{V}(X) = 0$, then there exists a constant c such that $\mathbb{P}(X = c) = 1$. That is, the only random variables with zero variance are constant with probability 1.

Finally, a result that applies to a sum of independent random variables. It is known as the *Bienaymé formula*, but is essentially just the Pythagorean Theorem.

Theorem 5.2.5 If $X_1, \ldots, X_n$ are pairwise independent random variables, then

$$\mathbb{V}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathbb{V}(X_i).$$

■ **Example 5.15** Take the binomial random variable $X \sim \text{Bin}(n, p)$. Then X is a sum of n i.i.d. Bernoulli random variables $\{X_i\}_{i=1}^{n}$, each with variance $p(1-p)$, so

$$\mathbb{V}(X) = \sum_{i=1}^{n} \mathbb{V}(X_i) = \sum_{i=1}^{n} p(1-p) = np(1-p).$$

The variance increases proportionally to the number of flips. ■

Putting some of the above facts together, we get some more details about the close relation between the MAD and standard deviation on one hand and vector norms on the other. Compare the following fact to the definition of a vector norm (Definition 3.4.1).

Fact 5.2.6 The standard deviation satisfies the following properties:

1. If $\sigma(X) = 0$, then there exists a constant c such that $\mathbb{P}(X = c) = 1$.
2. For any $a \in \mathbb{R}$ and random variable X, $\sigma(aX) = |a|\sigma(X)$.
3. For any random variables X and Y, $\sigma(X + Y) \leq \sigma(X) + \sigma(Y)$.

The MAD also satisfies the above properties.

5.3 Concentration Bounds

Generally, we would like to use the information from the mean and standard deviation to bound the probability that X deviates from its expected value by more than a certain amount. In other words, we would like some guarantee that the probability of an outlier occurring is rare, where what count as "outlier" and "rare" depend on the application. Here we cover two more formal inequalities that apply more generally, and one heuristic that applies specifically for normal distributions.

Theorem 5.3.1 — Markov's Inequality. For a *nonnegative* random variable X and $t > 0$,

$$\mathbb{P}(X \geq t) \leq \frac{\mathbb{E}[X]}{t}.$$

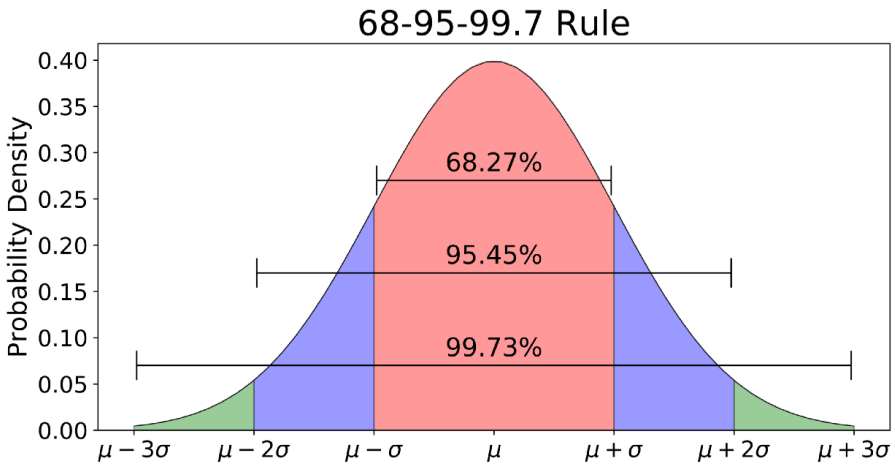


Figure 5.1: Illustration of the 68-95-99.7 rule. Source: [7]

Theorem 5.3.2 — Chebyshev's Inequality. Let X be a random variable with finite expected value $\mathbb{E}[X] = \mu$ and finite variance $\mathbb{V}(X) = \sigma^2$. Then for any $t > 0$,

$$\mathbb{P}(|X - \mu| \geq t\sigma) \leq \frac{1}{t^2}.$$

Chebyshev's inequality often gives loose bounds, but it is nonetheless able to provide the sorts of guarantees that we are after. For example, setting $t = 10$ in the above bound tells us that with at least 99% probability, a random variable will not differ from its mean by more than 10 standard deviations.

In the specific case of normal distributions, we can get bounds that are much stronger than Chebyshev's inequality. A common heuristic is known as the 68-95-99.7 rule.

Fact 5.3.3 — 68-95-99.7 Rule. If $X \sim \text{Normal}(\mu, \sigma^2)$, then

$$\begin{aligned}\mathbb{P}(|X - \mu| \leq \sigma) &\approx 0.68,\\ \mathbb{P}(|X - \mu| \leq 2\sigma) &\approx 0.95,\\ \mathbb{P}(|X - \mu| \leq 3\sigma) &\approx 0.997.\end{aligned}$$

The probabilities converge to 1 faster and faster the further out we go: one more standard deviation, for example, and we get $\mathbb{P}(|X - \mu| \leq 4\sigma) \approx 0.99994$. Thus, for a normal distribution, it is highly likely that X will take on a value within 3 or so standard deviations of its mean.

5.3.1 Weak Law of Large Numbers

One strong motivation for the definition of the expected value is that if we sample a random variable repeatedly, the average of the random samples will with high probability be close to the expected value. One version of this result is known as the Weak Law of Large Numbers (WLLN).

Theorem 5.3.4 — Weak Law of Large Numbers. Let $X_1, X_2, \ldots$ be a sequence of i.i.d. random

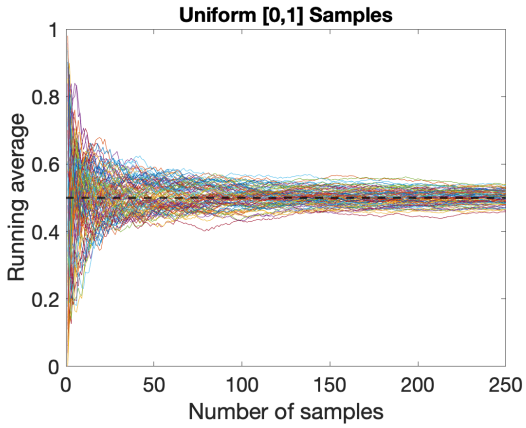


Figure 5.2: WLLN illustrated for repeated sampling from a Unif$[0,1]$ distribution.

variables, each with expected value μ. For each $n \geq 1$, define the *sample average*

$$\overline{X}_n = \frac{X_1 + \cdots + X_n}{n}.$$

Then for every $\varepsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}(|\overline{X}_n - \mu| > \varepsilon) = 0.$$

This theorem on its own does not tell us how large n needs to be to give any particular guarantee on the sampling error, just that the error will be small with probability converging to 1. The random variables do not need to have finite variance, although the theorem becomes simpler to prove if they do (e.g., using Chebyshev's inequality).

■ **Example 5.16** Figure 5.2 illustrates the WLLN for repeated sampling from a uniform $[0,1]$ distribution over 100 different trials. In this experiment, after 50 samples nearly all trials had an average between 0.4 and 0.6, and after 250 samples all trials had an average between 0.46 and 0.55. ■

Program 5.1: Weak Law of Large Numbers

```
rng(1)
trials = 100;
n = 250;
X = rand(n,trials);
X = cumsum(X)./(1:n)';
plot(X)
```

5.3.2 Central Limit Theorem

If the random variables being sampled also have finite variance, we can obtain a much more informative result about the long-run behavior.

Theorem 5.3.5 — Central Limit Theorem. Let $X_1, X_2, \ldots$ be a sequence of i.i.d. random variables, drawn from a distribution with expected value μ and finite variance $\sigma^2 > 0$. Then the sample averages, scaled appropriately, *converge in distribution* to the standard normal distribu-

tion.

$$\frac{\sqrt{n}}{\sigma}(\overline{X}_n - \mu) \xrightarrow{d} \text{Normal}(0,1).$$

This means that for any $z \in \mathbb{R}$,

$$\lim_{n\to\infty} \mathbb{P}\left[\frac{\sqrt{n}}{\sigma}(\overline{X}_n - \mu) \leq z\right] = \Phi(z)$$

Less formally, when n is sufficiently large the sample average $\overline{X}_n$ approximately has the normal distribution $\text{Normal}(\mu, \sigma^2/n)$. We note that we can derive without too much trouble the mean and standard deviation of the sample average, but the key result of the Central Limit Theorem is that the distribution also increasingly approximates the normal one.

Fact 5.3.6 For a sequence of random variables $X_1, \ldots, X_n \overset{i.i.d.}{\sim} X$, we have

$$\mathbb{E}[\overline{X}_n] = \mathbb{E}[X] \quad \text{and} \quad \mathbb{V}[\overline{X}_n] = \mathbb{V}[X]/n.$$

Proof. The result for the expected value holds by linearity of expectations:

$$\mathbb{E}[\overline{X}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[X_i] = \mathbb{E}[X].$$

The variances also add because the variables are assumed to be independent:

$$\mathbb{V}[\overline{X}_n] = \mathbb{V}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{V}[X_i] = \mathbb{V}[X]/n.$$

■

■ **Example 5.17** Consider the discrete random variable X with the PDF

$$p_X(x) = \begin{cases} 1/2 & x = 1, \\ 1/4 & x = 2, \\ 1/8 & x \in \{4, 12\}. \end{cases}$$

We sample from this distribution repeatedly, computing $\overline{X}_n$ for $n = 8, 20, 100$. Histograms of the sample means over 10,000 trials are shown in Figure 5.3. With an average over only 8 samples, the distribution is not very closely concentrated around the mean and does not look anything like a normal distribution. With 20 samples the distribution is closer to normal, but still a little asymmetric. By the time we hit 100 samples, the approximation looks pretty good. Note that the approximation can never be exact—the normal distribution is nonzero on the entire real number line, but the sample average necessarily lies between 1 and 12. ■

The Central Limit Theorem gives us a nice way to simplify probability problems. When we want to analyze the average behavior of a random variable X over a large number of samples, we generally don't have to know exactly what the distribution of $\overline{X}_n$ looks like or even the distribution of X. Instead, we can approximate $\overline{X}_n$ with a normal distribution and do our calculations using the standard normal CDF Φ. Thus as long as we can evaluate Φ reliably, we can get decent approximations for a wide variety of other distribution functions.

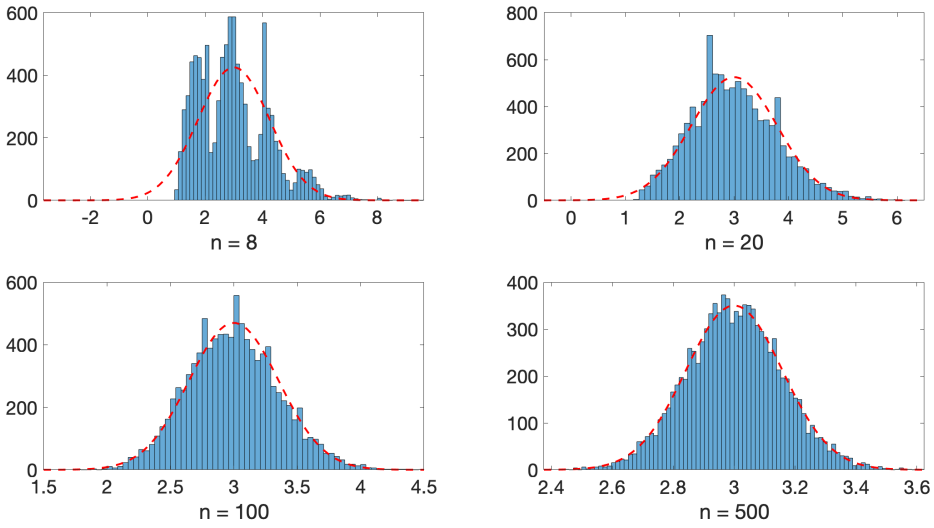


Figure 5.3: Demonstration of Central Limit Theorem on a discrete distribution

Program 5.2: Central Limit Theorem Example

```
rng(1)
figure('position',[200 200 1200 600])
vals = [1, 2, 4, 12];
wts = [1/2, 1/4, 1/8, 1/8];
mu = dot(vals,wts);
sigma = std(vals,wts);
ns = [8, 20, 100, 500];
bws = [2/15, 5/48, 1/24, 1/72];
trials = 10000;

for i = 1:4
    n = ns(i);
    subplot(2,2,i)

    X = randsample(vals,n*trials,true,wts);
    X = reshape(X,n,trials);
    Xbar = sum(X)/n;

    h = histogram(Xbar); hold on
    bw = bws(i);
    h.BinWidth = bw;

    m = max(abs(Xbar-mu));
    xs = linspace(mu -m, mu + m);
    ys = trials*bw*normpdf(xs,mu,sigma/sqrt(n));
    plot(xs,ys,'r--','linewidth',2)
    xlim([mu-m,mu+m])
    xlabel(sprintf("n = %d",n), 'fontsize',20)
    ax = gca;
    ax.FontSize = 20;
end
```

■ **Example 5.18** Let $X \sim \text{Unif}[0,1]$. If we take the average over $n = 100$ i.i.d. samples, what is the probability that $45 \leq \sum_{i=1}^{100} X_i \leq 51$?

The expected value of X is 1/2, and by Example 5.10 the standard deviation is $\sigma = \sqrt{1/12} \approx 0.2887$. Therefore,

$$\begin{aligned}
\mathbb{P}\left[45 \leq \sum_{i=1}^{100} X_i \leq 51\right] &= \mathbb{P}\left[0.45 \leq \overline{X}_{100} \leq 0.51\right] \\
&= \mathbb{P}\left[-0.05 \leq \overline{X}_{100} - \mu \leq 0.01\right] \\
&= \mathbb{P}\left[-\sqrt{3} \leq \frac{\sqrt{n}}{\sigma}(\overline{X}_{100} - \mu) \leq \sqrt{0.12}\right] \\
&\overset{\text{CLT}}{\approx} \Phi(\sqrt{0.12}) - \Phi(-\sqrt{3}) \\
&\approx 0.594.
\end{aligned}$$

■

■ **Example 5.19** A standard wager in craps is an even bet where the player has a $p = 244/495 \approx 0.493$ chance of winning. About how many games must be played in order for the casino to have at least a 99% chance of turning a profit?

Let $X_1, \ldots, X_n \sim \text{Ber}(p)$, and let $\overline{X}_n$ be the sample average. Then the expected value is $\mu = p$ and the standard deviation $\sigma = \sqrt{p(1-p)} \approx 0.49995$. The house turns a profit if the player wins less than half of the games played, so we want to find the value of n such that

$$\mathbb{P}(\overline{X}_n < 0.5) \geq 0.99$$

Using the Central Limit Theorem, we find that

$$\begin{aligned}
\mathbb{P}(\overline{X}_n < 0.5) &\approx \mathbb{P}\left(\frac{\sqrt{n}}{\sigma}(\overline{X}_n - \mu) < (0.5-p)\sqrt{\frac{n}{p(1-p)}}\right) \\
&\overset{\text{CLT}}{\approx} \Phi\left((0.5-p)\sqrt{\frac{n}{p(1-p)}}\right).
\end{aligned}$$

The approximate number of games needed is then found by solving

$$(0.5-p)\sqrt{\frac{n}{p(1-p)}} \geq \Phi^{-1}(0.99),$$

which gives (to the nearest integer) the answer $n \geq 27,057$.

Checking this answer in Matlab gives `binocdf(n/2,n,p)` ≈ 0.989998, so we weren't too far off. Using $n \geq 27,061$ suffices when n is odd. When n is even, the number must be a little larger because ties are possible. It can be tricky to use continuous approximations for discrete problems! ■

R Why do so many measured quantities in the real world, such as heights[1], approximately have a normal distribution? The more general principle is this: if a single measured value is the weighted average of many small effects, then that value should have a distribution that is approximately normal.

[1] At least within a single gender. Heights over the total population would look more like a mixture of two normal distributions.

5.3.3 Continuity Correction

When a random variable X has a binomial distribution, we have some flexibility in how exactly we compare the distribution of X to the standard normal CDF Φ. In particular, if $X \sim \text{Bin}(n,p)$, then for any integer k it holds that

$$\mathbb{P}(X \leq k) = \mathbb{P}(X < k+1)$$

because X does not take on fractional values. So when we compare the distribution of X to the normal distribution, should we make the point of comparison k, $k+1$, or somewhere in between?

If $np(1-p)$ is sufficiently large, it turns out that it is fairly reasonable to use the approximation

$$\mathbb{P}(X \leq k) \approx \mathbb{P}(Z \leq k + \frac{1}{2}),$$

where Z is a normal random variable with the same mean and standard deviation as X.

■ **Example 5.20** Flip 100 fair coins. What is the probability of getting *exactly* 50 heads?

The variable $X \sim \text{Bin}(100,50)$ has mean 50 and standard deviation $\sigma = \sqrt{np(1-p)} = 5$, so let $Z \sim \text{Normal}(50, 5^2)$. The approximation

$$\mathbb{P}(50 \leq X \leq 50) \approx \mathbb{P}(50 \leq Z \leq 50)$$

would not get us very far because the latter quantity is equal to zero! So we make a continuity correction:

$$\begin{aligned}
\mathbb{P}(50 \leq X \leq 50) &= \mathbb{P}(49.5 \leq X \leq 50.5) \\
&\overset{\text{CLT}}{\approx} \mathbb{P}(49.5 \leq Z \leq 50.5) \\
&= \mathbb{P}(-0.1 \leq (Z-50)/5 \leq 0.1) \\
&= \Phi(0.1) - \Phi(-0.1) \\
&\approx 0.07966.
\end{aligned}$$

This answer makes use of Theorem 5.2.3, which implies that if $Z \sim \text{Normal}(50, 5^2)$ then $(Z-50)/5 \sim \text{Normal}(0,1)$. The true answer, given by `binopdf(50,100,0.5)`, is about 0.07959, so the continuity correction worked pretty well in this scenario! ■

■ **Example 5.21** A call center has 90 workers. To save time, a centralized system calls 800 people at once and assigns a worker to anyone who answers the phone. If only 10 percent landline calls get answered on average, what is the approximate probability that more than 100 people will pick up (thus overloading the workers)?

Model the number of people answering the phone by a Bernoulli random variable $X \sim \text{Ber}(800, 0.1)$. Then the mean of X is $\mu = 90$ and the standard deviation $\sigma = \sqrt{np(1-p)} \approx 8.485$. From there, we let $Z \sim \text{Normal}(\mu, \sigma^2)$ and find that

$$\begin{aligned}
\mathbb{P}(X > 100) &= \mathbb{P}(X \geq 100.5) \\
&\overset{\text{CLT}}{\approx} \mathbb{P}(Z \geq 100.5) \\
&\approx \mathbb{P}((Z-\mu)/\sigma \geq 1.2374) \\
&= 1 - \Phi(1.2374) \\
&\approx 0.1080.
\end{aligned}$$

Without the continuity correction, we get a probability of about 0.1193. The true answer is about 0.1094, so the continuity correction improved over the initial strategy. ■

5.4 Matlab Commands

Summary Statistics

- `mean(X)`: the mean of the elements in the vector `X`. See also `median`, `mode`.
- `min(X)`: the minimum element in `X`. Using `[m,ix] = min(X)` returns the minimum value `m` and the index `i` at which it is found. See also `max`.
- `mink(X,k)`: returns the smallest `k` elements in `X`. See also `maxk`.
- `var(X)`: returns an unbiased estimate of the variance, where `X` is a vector of samples from a distribution. `var(X) = var(X,0)` normalizes by `N-1` and `var(X,1)` normalizes by `N`.
- `var(X,w)`: returns the variance of the values `X` with weights given by `w`.
- `std(X)`: returns the standard deviation. Same usage as `var(X)`, but `std(X)` is not an unbiased estimator of the standard deviation.

6. Monte Carlo Integration

The general idea of *Monte Carlo methods* is to use randomization to estimate the answer to a problem that we may not be able to solve deterministically. We focus in particular on the problem of computing an integral.

Monte Carlo integration combines two key tools that we have already encountered. The first is the Law of Large Numbers (Theorem 5.3.4): that if we repeatedly sample a random variable $X_1, X_2, \ldots$ with expectation μ, then the sample average $\overline{X}_n$ satisfies

$$\lim_{n\to\infty} \mathbb{P}(|\overline{X}_n - \mu| > \varepsilon) = 0$$

for any fixed $\varepsilon > 0$. The second tool is the Law of the Unconscious Statistician (Theorem 5.1.5): in the continuous case, if X has PDF f_X, then for any function g it holds that

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x)\, dx.$$

This theorem was originally presented with the notion that we could compute an expected value by solving a related integral. The key idea behind Monte Carlo integration is that we can use it in the opposite direction as well—in order to solve an integral, we can estimate an expected value!

6.1 The Basic Strategy

Technique 6.1.1 — Monte Carlo Integration. In order to estimate the integral

$$I = \int_{-\infty}^{\infty} g(x) f(x)\, dx,$$

repeatedly sample a random variable $X_1, X_2, \ldots, X_n$ with PDF f, and return

$$\bar{I}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i).$$

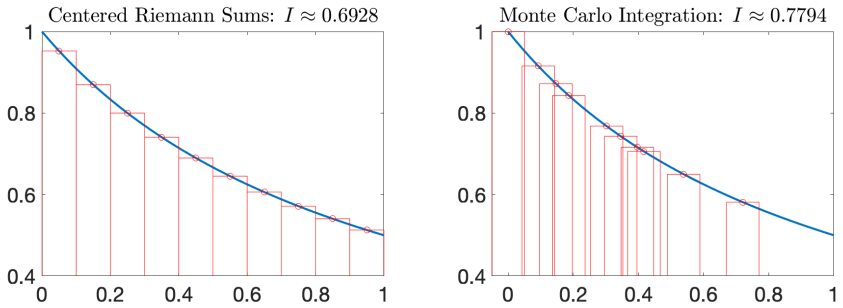


Figure 6.1: Two approximations of $\ln(2) \approx 0.6931$. Left: Centered Riemann sums. Right: Monte Carlo integration.

When the domain is a closed interval, the simplest strategy in this scenario is to sample from a uniform distribution. In order to estimate the integral

$$I = \int_a^b g(x)\,dx,$$

we can repeatedly sample $X_1, \ldots, X_n \overset{i.i.d.}{\sim} \text{Unif}[a,b]$ and return

$$\bar{I}_n = \frac{b-a}{n} \sum_{i=1}^{n} g(X_i).$$

This works because the PDF of the uniform distribution is $\frac{1}{b-a}$ on $[a,b]$ and zero elsewhere.

■ **Example 6.1** Consider the problem of estimating

$$\ln(2) = \int_0^1 \frac{1}{1+x}\,dx.$$

We estimate the integral as $\ln(2) \approx \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1+X_i}$, where $X_1, \ldots, X_n \overset{i.i.d.}{\sim} \text{Unif}[0,1]$. ■

Figure 6.1 shows a side-by-side comparison of using centered Riemann sums to estimate the integral versus our Monte Carlo technique with $n = 10$ samples. Under this visualization, Monte Carlo integration is something like taking a Riemann sum where the sampled points are chosen randomly instead of being evenly spaced.

With $n = 10$ samples, the centered Riemann sum is far superior. It gives an approximation with a relative error of about $4.5 \cdot 10^{-4}$, while in this particular trial the Monte Carlo approximation had a relative error of 0.12.

6.1.1 Approximating Pi

One classic application of Monte Carlo sampling is finding the area of a circle by sampling points uniformly at random from the enclosing square. Suppose we have a circle E of radius 1, and therefore area $\pi r^2 = \pi$, enclosed in the square $\Omega = [-1,1]^2$. If we sample points uniformly at random from the square, then the probability that such a point will fall inside the circle is

$$\mathbb{P}(E) = \frac{\text{area}(E)}{\text{area}(\Omega)} = \frac{\pi}{4} \approx 0.785.$$

We can therefore approximate the area of the circle by randomly sampling points from Ω and observing what proportion of those points fall within the circle. This problem can be cast in terms

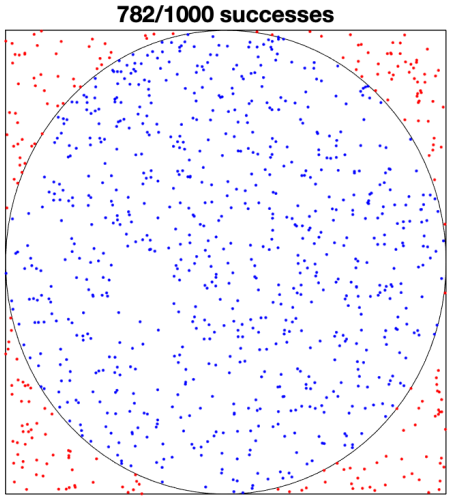


Figure 6.2: Results of 1000 Monte Carlo trials for approximating pi.

of an integral by using the indicator function $\mathbf{1}_E$, which takes on the value 1 if $\omega \in E$ and zero otherwise, and using the identity $\mathbb{P}(E) = \mathbb{E}[\mathbf{1}_E]$ (Example 5.6).

$$\frac{\pi}{4} = \mathbb{P}(E) = \mathbb{E}[\mathbf{1}_E] = \int_{x=-1}^{1} \int_{y=-1}^{1} \mathbf{1}_E(x,y) f_{X,Y}(x,y)\, dy\, dx,$$

where $f_{X,Y}$ is the uniform distribution on $[-1,1]^2$. We can therefore sample $X, Y \overset{i.i.d.}{\sim} \text{Unif}[-1,1]$ and approximate pi as

$$\pi \approx \frac{4}{n} \sum_{i=1}^{n} \mathbf{1}_E(X_i, Y_i).$$

Program 6.1 presents sample code for this approach, and results for one trial with $n = 1000$ are shown in Figure 6.2. The success rate of 782/1000 translates to the approximation $\pi \approx 3.128$.

Program 6.1: Finding the area of a circle

```
rng(1)
n = 1000;
S = 0;
for i = 1:n
    x = 2*rand-1;
    y = 2*rand-1;
    if (x^2+y^2 ≤ 1)
        S = S + 1;
    end
end
```

6.1.2 Motivation

Given the results of Example 6.1, why should we even bother with Monte Carlo techniques? Even for a technique as simple as the centered Riemann sum (never mind the trapezoid rule or more sophisticated rules!), the error decreases proportionally to $1/n^2$. With Monte Carlo integration,

it decreases proportionally to $1/\sqrt{n}$. At a glance, it seems that Monte Carlo integration is highly uncompetitive.

If we stuck to working in one dimension, this impression would be correct: Monte Carlo algorithms are not even remotely competitive for estimating low-dimensional integrals. But when we begin to work in higher dimensions, Monte Carlo methods start to become more attractive. Consider the *Borehole function* [20]

$$f(\text{inputs}) = \frac{2\pi T_u(H_u - H_l)}{\ln(r/r_w)\left(1 + \frac{2LT_u}{\ln(r/r_w)r_w^2 K_w} + \frac{T_u}{T_l}\right)},$$

a function that models the water flow through a borehole. It has 8 dimensions with the inputs

$r_w \in [0.0, 0.15]$	radius of borehole (m)
$r \in [100, 50000]$	radius of influence (m)
$T_u \in [63070, 115600]$	transmissivity of upper aquifer (m^2/yr)
$H_u \in [990, 1110]$	potentiometric head of upper aquifer (m)
$T_l \in [63.1, 116]$	transmissivity of lower aquifer (m^2/yr)
$H_l \in [700, 820]$	potentiometric head of lower aquifer (m)
$L \in [1120, 1680]$	length of borehole (m)
$K_w \in [9855, 12045]$	hydraulic conductivity of borehole (m/yr)

If we want to estimate the integral of f over its domain with 10 points in each direction, then we will need to evaluate $f(\mathbf{x})$ at 10^8 points in the cube. A personal computer could still reasonably handle a problem of this size, but for a model with $d = 15$ or $d = 20$ different inputs the cost will become prohibitive. The hope with Monte Carlo methods is that if the value of the function does not vary too much over its domain, then a small number of randomly chosen samples will be able to give us a decent estimate of the integral.

6.2 Error Estimation

Given a desired error threshold, we would like to be able to determine the number of samples needed to ensure that with a certain probability, the approximation error of our estimate falls below that threshold. If we know the standard deviation of the quantity we are sampling, then we can solve this problem with the aid of the Central Limit Theorem.

Definition 6.2.1 Let $\overline{X}_n$ be a sample average from a distribution with standard deviation σ. The *standard error on the mean* is given by

$$\sigma_{\overline{X}_n} = \frac{\sigma}{\sqrt{n}}.$$

The standard error is equal to the standard deviation of $\overline{X}_n$, and represents the average distance between $\overline{X}_n$ and its expectation μ. By the Central Limit Theorem, for sufficiently large n the sample average $\overline{X}_n$ will approximately have the distribution Normal$(\mu, \sigma^2_{\overline{X}_n})$, and therefore

$$\mathbb{P}(\mu - t\sigma_{\overline{X}_n} \leq \overline{X}_n \leq \mu + t\sigma_{\overline{X}_n}) \approx \Phi(t) - \Phi(-t).$$

By rearranging the inequalities, we find that the above approximation is equivalent to

$$\mathbb{P}(\overline{X}_n - t\sigma_{\overline{X}_n} \leq \mu \leq \overline{X}_n + t\sigma_{\overline{X}_n}) \approx \Phi(t) - \Phi(-t).$$

By setting t to some particular value, we obtain a *confidence interval* for μ.

Definition 6.2.2 For random variables $X_1, \ldots, X_n \overset{i.i.d.}{\sim} X$ with standard deviation σ, the *exact 95% confidence interval* is the interval $[\overline{X}_n - 1.96\sigma_{\overline{X}_n}, \overline{X}_n + 1.96\sigma_{\overline{X}_n}]$.

One way to interpret the confidence interval is that if we conduct an experiment many times, then the true mean μ will fall within the 95% confidence interval for roughly 95% of the experiments. It does *not* mean that there is a 95% chance that the mean falls lies within any one specific interval! At least under one mode of interpretation, an interval, once observed, either contains the mean or it doesn't, and probability no longer factors into it. Nonetheless, confidence intervals are frequently used as an estimate of plausible values for the unknown parameter μ.

6.2.1 Sample Variance

In practice, we often will not know the standard error and must estimate it instead.

Definition 6.2.3 For random variables $X_1, \ldots, X_n \overset{i.i.d.}{\sim} X$ with sample average $\overline{X}_n$, the *unbiased sample variance* is

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X}_n)^2$$

The use of the factor $n-1$ rather than n is known as *Bessel's correction*, and ensures that the estimator is in fact unbiased:

Theorem 6.2.1 For random variables $X_1, \ldots, X_n \overset{i.i.d.}{\sim} X$ with variance σ^2, the unbiased sample variance satisfies

$$\mathbb{E}[S^2] = \sigma^2.$$

However, the biased sample variance, using factor n rather than $n-1$, is closer on "average" to the true underlying standard deviation.

Following close behind are the sample standard deviation and sample standard error. We will use the versions derived from the unbiased sample variance as a default, although these estimators are not themselves unbiased.

Definition 6.2.4 For random variables $X_1, \ldots, X_n \overset{i.i.d.}{\sim} X$, the *sample standard deviation* is

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X}_n)^2}$$

and the *sample standard error* is

$$S_{\overline{X}_n} = \mathrm{SE} = \frac{S}{\sqrt{n}}.$$

We can therefore generate a confidence interval by using the sample standard error if the true value is not known.

Definition 6.2.5 For random variables $X_1, \ldots, X_n \overset{i.i.d.}{\sim} X$ with standard deviation σ, the 95% *confidence interval* is the interval $[\overline{X}_n - 1.96S_{\overline{X}_n}, \overline{X}_n + 1.96S_{\overline{X}_n}]$.

Up to this point, all of the definitions have been purely in terms of random variables. When working with sampled data, we will use lowercase letters in order to distinguish them from the random variables describing their distribution.

Definition 6.2.6 A *realization* x of a random variable X is an element of $X(\Omega)$. A *random sample* $\{x_1,\ldots,x_n\}$ is a set of realizations of n i.i.d. random variables.

Notation 6.1. *Given a random sample $x_1,\ldots,x_n$ from a distribution X, the sample mean, sample variance, sample standard error, and 95% confidence interval are given by*

- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$,
- $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2$,
- $s_{\bar{x}} = s/\sqrt{n}$,
- $CI = [\bar{x} - 1.96 s_{\bar{x}},\ \bar{x} + 1.96 s_{\bar{x}}]$.

■ **Example 6.2** We flip a coin with unknown bias 100 times, getting 60 heads. Assuming that the data is a random sample from $\mathrm{Ber}(p)$ with unknown p, we find that $\bar{x} = 0.6$, $s^2 = 24/99 \approx 0.242$, and $s_{\bar{x}} \approx 0.0492$. The 95% confidence interval is therefore given by

$$CI \approx [0.503,\ 0.697].$$

Since this interval excludes the value 0.5, is it reasonable to conclude that this coin is probably biased? Not necessarily! It is awfully hard to bias a coin in practice, particularly if you catch the flips in your hand. If you give weight to this prior knowledge, it seems much more reasonable to conclude that the outcome if the experiment was a fluke rather than that something is wrong with the coin. ■

■ **Example 6.3** Using Monte Carlo integration to estimate

$$\ln(2) = \int_0^1 \frac{1}{1+x}\,dx,$$

about how many samples will we need to get two digits of accuracy?

To answer this, we will use the exact mean and standard deviation. Let $X \sim \mathrm{Unif}[0,1]$, and let $g(x) = 1/(1+x)$. Then $\mathbb{E}[g(X)] = \ln(2) \approx 0.693$, and

$$\mathbb{V}(g(X)) = \mathbb{E}[g^2(X)] - \ln(2)^2 = \int_0^1 \frac{1}{(1+x)^2}\,dx - \ln(2)^2 = \frac{1}{2} - \ln(2)^2 \approx 0.0195.$$

The standard error is therefore given by

$$\mathrm{SE} = \frac{\sqrt{1/2 - \ln(2)^2}}{\sqrt{n}} \approx 0.1398/\sqrt{n}.$$

Two digits of accuracy corresponds roughly to a relative error of 0.05. We would like the radius of the 95% confidence interval to be less than this quantity, which corresponds to solving the inequality

$$1.96 \cdot \mathrm{SE} \leq 0.05\mu,$$

which has the solution $n \geq 63$. The exact confidence interval therefore suggests that if we take 63 samples, then 95 percent of the time the relative error of our solution will be less than 0.05. Running the code in Program 6.2 verifies this prediction empirically, with the relative error falling within the desired bounds 9,492 times out of 10,000. Figure 6.3 illustrates the results in the form of a histogram. ■

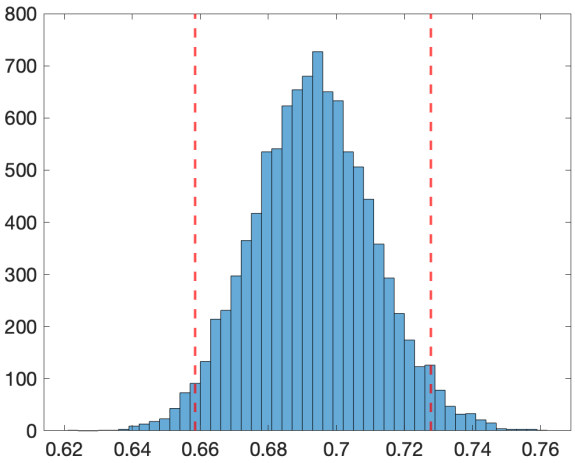


Figure 6.3: Histogram of Monte Carlo estimates for $\ln(2)$. Dashed red lines show a relative error of 0.05.

Program 6.2: Approximating $\log(2)$ with Monte Carlo integration

```
rng(1)
n = 63;
g = @(x)1./(1+x);
mu = log(2);
ntrials = 10000;
ms = zeros(1,ntrials);

for i = 1:ntrials
    x = rand(1,n);
    mx = mean(g(x));
    ms(i) = mx;
end

inBounds = sum(abs(ms-mu)/mu < 0.05);
histogram(ms), hold on
xline(mu*0.95,'r--'),
xline(mu*1.05,'r--')
```

6.3 Variance Reduction

We saw in Example 6.3 that the variance of Monte Carlo integration, and therefore the number of samples required to reach a certain level of accuracy, depends on the variance $\mathbb{V}(g(X))$. This means that the efficacy of Monte Carlo techniques can depend on the function to be integrated: if $g(x)$ is nearly constant then the variance will be small, but if the value of $g(x)$ fluctuates wildly then our Monte Carlo techniques will not perform as well. Here we cover a few strategies for handling functions with high variance.

6.3.1 Control Variates

Suppose that we would like to approximate $\mathbb{E}[X]$ by sampling X, and we have access to a variable Y which is related to X and whose expected value $\mathbb{E}[Y] = \mu_Y$ is known. We will define exactly what we mean by "related" in a later chapter, but for now we will take it to mean that $|X(\omega) - Y(\omega)|$ is

generally small compared to $X(\omega)$. We then use the linearity of expectation and to deduce that

$$\mathbb{E}[X] = \mathbb{E}[X-Y] + \mathbb{E}[Y] = \mathbb{E}[X-Y] + \mu_Y.$$

The idea is that if X and Y are closely related, then $\mathbb{V}(X-Y)$ should be substantially smaller than $\mathbb{V}(X)$. Thus by sampling $(X-Y)$ and adding μ_Y as a correction term, we can estimate μ_X much more efficiently than by sampling X directly.

Technique 6.3.1 — Control Variates. If we can sample from related distributions X and Y and have access to μ_Y, then sample from $X-Y$ and approximate $\mu_X \approx \overline{(X-Y)}_n + \mu_Y$.

It is crucial to note that the variables X and Y cannot be independent for this technique to work. If they were independent, then the Bienaymé formula (Theorem 5.2.5) implies that $\mathbb{V}(X-Y) = \mathbb{V}(X) + \mathbb{V}(Y)$, which defeats the purpose of including Y. This also means that X and Y must be sampled jointly rather than independently. That is, we must find $(X-Y)(\omega)$ for a common ω, not $X(\omega_1) - Y(\omega_2)$ where ω_1 and ω_2 are drawn separately. It is also possible to choose a Y such that $\mathbb{V}(X-Y)$ is greater than $\mathbb{V}(X)$, so not just any variable Y will do. We will revisit the problem of how to choose Y in a later chapter.

■ **Example 6.4** We observe that on the interval $[0,1]$, the function $g(x) = \frac{1}{1+x}$ has behavior somewhat similar to the function $h(x) = 1 - \frac{x}{2}$ (Figure 6.4). It is simple to check that $\int_0^1 h(x)\,dx = 3/4$, so we approximate $\ln(2)$ by sampling $X \sim \text{Unif}[0,1]$ and using the estimate

$$\ln(2) \approx \frac{3}{4} + \frac{1}{n}\sum_{i=1}^{n} g(X_i) - h(X_i).$$

Code is presented in Program 6.3. Reporting the confidence intervals in the form $\bar{x} \pm 1.96 s_{\bar{x}}$, we find the original and variance-reduced methods respectively give the estimates

$$\ln(2) \approx 0.701 \pm 0.029,$$
$$\ln(2) \approx 0.694 \pm 0.005.$$

The true value of approximately 0.693 falls within both confidence intervals, but the second interval is less than 1/5 the width of the first. Thus using a control variate has substantially improved the estimate. ■

Program 6.3: Monte Carlo with Control Variates

```
rng(1)
g = @(x)1./(1+x);
h = @(x)1-x/2; % expected value is 0.75

n = 100;
x = rand(1,n);
y1 = g(x);
y2 = g(x)-h(x) + 0.75; % using the same random samples x!
se1 = std(y1)/sqrt(n);
se2 = std(y2)/sqrt(n);
```

6.3.2 Importance Sampling

The second strategy involves carefully choosing the probability distribution from which X will be sampled. Using the Law of the Unconscious Statistician (Theorem 5.1.5), we observe that in

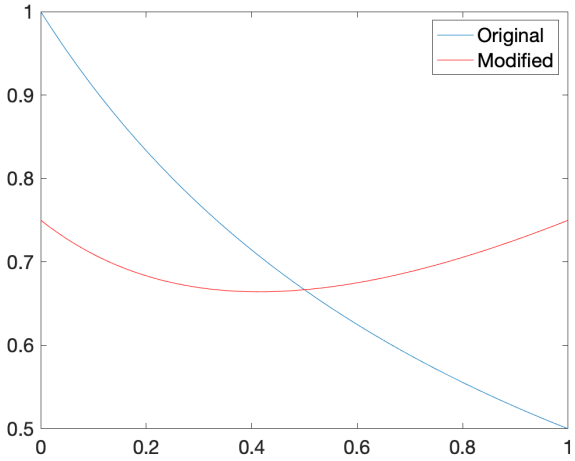


Figure 6.4: Two functions have the same mean, but the second has significantly smaller variance.

general we have

$$\int g(x)\,dx = \int \frac{g(x)}{f(x)} f(x)\,dx = \mathbb{E}[g(X)],$$

where X is a random variable with PDF $f(x)$ So assuming we have the ability to sample from a wide variety of distributions, we would do well to choose the one that minimizes the variance $\mathbb{V}(g(X)/f(X))$.

This gives us a few simultaneous goals:

1. $f(x) > 0$ whenever $g(x) > 0$,
2. f is roughly proportional to g on the domain of interest,
3. It is easy to compute the density f and to sample from it.

When the integral is over a finite interval, making f the uniform distribution satisfies the third criterion but not necessarily the first two. The ideal function for the first two criteria would be $f(x) = \alpha \cdot g(x)$, but this quickly runs into a problem: since $f(x)$ must be a valid PDF, we need to set α so that

$$1 = \int f(x)\,dx = \alpha \int g(x)\,dx,$$

which means that $\alpha = (\int g(x)\,dx)^{-1}$. So to find the proper value of α, we need to solve the integral that we wished to estimate in the first place!

■ **Example 6.5** Revisit the problem of estimating $\log(2)$ by integrating $g(x) = 1/(1+x)$, this time using importance sampling. We will take the density function $f(x) = \frac{4}{3}(1 - x/2)$ for $x \in [0,1]$ and zero elsewhere. Sampling from this distribution is non-trivial, but we can do so by generating $U \sim \text{Unif}[0,1]$ and taking $X = 2 - \sqrt{4-3U}$ (more on this sort of sampling in the next chapter). With this strategy, we get the approximation

$$\ln(2) \approx 0.694 \pm 0.005,$$

which is more or less the same improvement that we got by using a control variate. ■

Importance sampling becomes even more relevant when the domain of integration is unbounded. Uniform sampling is out of the question, so we have to decide more carefully what probability distribution we would like to sample from.

6.3.3 Stratification

A third strategy, stratification, is useful when the function $g(x)$ varies much more over some parts of the domain than others. Assuming for simplicity that we have a bounded domain of integration D, the idea is to split the domain into m disjoint pieces $D = \bigcup_{i=1}^{m} D_i$, and use the relation

$$\int_D g(x)\,dx = \sum_{i=1}^{m} \int_{D_i} g(x)\,dx = \sum_{i=1}^{m} |D_i| \mathbb{E}[g(X^{(i)})],$$

where $|D_i|$ is the area of the domain D_i and X_i is distributed uniformly on D_i. We can then sample each expectation independently. Using n_i samples for domain i, we get the approximation

$$\int_D g(x)\,dx \approx \sum_{i=1}^{m} \frac{|D_i|}{n_i} \sum_{j=1}^{n_i} g(X_j^{(i)}).$$

Proportionate Sampling

One simple strategy choose the sampling numbers proportionally to the size of their piece of the domain: $n_i = \frac{n|D_i|}{|D|}$. Essentially, this guarantees that we sample from the domain with the same density as before, but make sure that our samples are spread a little more uniformly than they might be otherwise. The variance of the estimator is then

$$\begin{aligned}
\mathbb{V}\left(\sum_{i=1}^{m} \frac{|D_i|}{n_i} \sum_{j=1}^{n_i} g(X_j^{(i)})\right) &= \sum_{i=1}^{m} \frac{|D_i|^2}{n_i^2} \sum_{j=1}^{n_i} \mathbb{V}(g(X_j^{(i)})) \\
&= \sum_{i=1}^{m} \frac{|D|}{n} |D_i| \mathbb{V}(g(X^{(i)})) \\
&= \frac{|D|}{n} \sum_{i=1}^{m} \int_{D_i} (g(x) - \mu_i)^2\,dx \\
&\leq \frac{|D|}{n} \sum_{i=1}^{m} \int_{D_i} (g(x) - \mu)^2\,dx \\
&= \frac{|D|}{n} \int_{D} (g(x) - \mu)^2\,dx \\
&= \mathbb{V}\left(\frac{|D|}{n} \sum_{j=1}^{n} g(X_j)\right).
\end{aligned}$$

Here $\mu = \mathbb{E}[g(X)]$ where $X \sim \text{Unif}(D)$ and $\mu_i = \mathbb{E}[g(X^{(i)})]$, where $X_i \sim \text{Unif}(D_i)$. The inequality holds because in general $\mathbb{E}[(X-c)^2]$ is minimized by setting $c = \mathbb{E}[X]$. This shows that the simple sampling strategy is always at least as good as non-stratified sampling.

Optimal Sampling

A more sophisticated strategy is to choose the sampling numbers n_i so that areas with greater variance get sampled more often. Given a total number of samples $n = \sum_{i=1}^{m} n_i$ and allowing fractional numbers of samples to simplify the analysis, it turns out that the optimal strategy is to set

$$\begin{aligned}
n_i &= \lambda |D_i| \sigma_i, \quad \text{where} \\
\lambda &= \frac{n}{\sum_{i=1}^{m} |D_i| \sigma_i}, \\
\sigma_i &= \sqrt{\mathbb{V}(g(X^{(i)}))}.
\end{aligned}$$

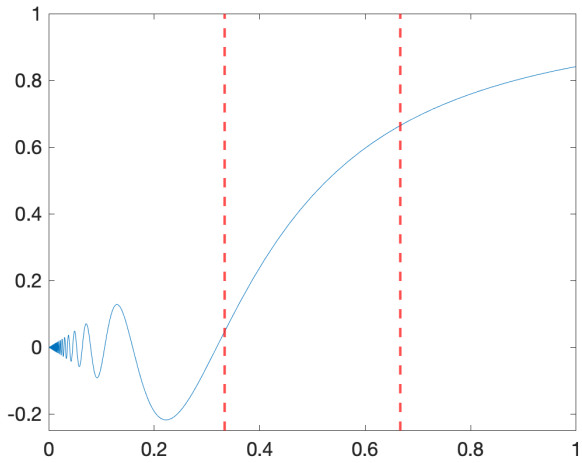


Figure 6.5: Integrating $x\sin(1/x)$ by dividing the domain $[0,1]$ into 3 equal parts.

The resulting variances of the estimators using optimal, proportionate, and non-stratified sampling are respectively

$$\underbrace{\frac{1}{n}\left(\sum_{i=1}^{m}|D_i|\sigma_i\right)^2}_{\text{optimal}} \leq \underbrace{\frac{|D|}{n}\sum_{i=1}^{m}|D_i|\sigma_i^2}_{\text{proportionate}} \leq \underbrace{\frac{|D|^2}{n}\sigma^2}_{\text{regular}}.$$

The first inequality can be shown using a judicious application of the Cauchy-Schwarz inequality (Theorem 3.4.3), and the second was proved in the previous section.

■ **Example 6.6** Consider using Monte Carlo techniques to approximate the integral

$$\int_0^1 x\sin(1/x)\,dx \approx 0.37853.$$

We break the interval $[0,1]$ into even thirds. With the basic approach, we take 300 samples uniformly at random. With the second, we take 100 from each subinterval. With the third, we use sampling numbers $\{86,169,45\}$, since the middle interval has the greatest variance and the rightmost interval has the least. We end up with the approximations

$$\int_0^1 x\sin(1/x)\,dx \approx 0.400 \pm 0.040,$$
$$\int_0^1 x\sin(1/x)\,dx \approx 0.373 \pm 0.014,$$
$$\int_0^1 x\sin(1/x)\,dx \approx 0.380 \pm 0.012.$$

Thus the "optimal" sampling strategy was marginally better than proportional sampling, and both were substantially better than the basic method. ■

6.4 Monte Carlo Simulation (TODO)

Put a wider variety of applications for Monte Carlo methods here.

- Nagel–Schreckenberg traffic model
- Geometric Brownian motion and stock prices

6.5 Quasirandom Sampling (TODO)

Possibly mention Sobol sequences. The hope is to space the points more evenly in order to get convergence closer to $1/N$ than $1/\sqrt{N}$.

6.6 Matlab Commands

Integration

- `@`: Used to create a `function_handle`. When a function accepts another function as an input, the input must be in the form of a `function_handle`. Example: `f = @cos`, or `g = @(x) 2*x`. It is also generally a good idea to design these functions to allow vectorized operations, such as `f = @(x)x.^2` rather than `f = @(x)x^2`.
- `q = integral(f,a,b)`: Numerically evaluates the integral of `f` on `[a,b]`. The input `f` should be in the form of a `function_handle`. See also `integral2` and `integral3` for numerical integration in 2 and 3 dimensions.
- `z = trapz(x,y)`: Estimates an integral using the trapezoid method, with x-values and y-values given by `x` and `y`. Using `z = trapz(y)` assumes unit spacing.
- `z = cumtrapz(x,y)`: Returns a vector containing the cumulative integral. Using `z = cumtrapz(y)` assumes unit spacing.

7. Random Number Generation

Through all of our talk about random variables and Monte Carlo methods, we tacitly assumed that we could generate random numbers from a given distribution (mostly the uniform distribution) at will. Here, we take a look at how this is done in practice. Large portions of this chapter are taken from [17].

7.1 Pseudorandom Numbers

Start a fresh session of Matlab, set `format long`, type `rand`, and you will get the number

```
0.814723686393179.
```

Matlab's "random" number generation procedure is deterministic. Given that computers behave deterministically (at least in principle), this should perhaps not be too much of a surprise. It is possible to generate random numbers by mechanical means – simple methods include rolling a die or spinning a roulette wheel, but these are too slow to be useful for many modern applications. More sophisticated methods involve measuring radioactive decay or radio noise[9], but most computers will settle for *pseudorandom number generators* (PRNGs). One definition was given in 1951 by Berkeley professor D. H. Lehmer [17]:

> A random sequence is a vague notion... in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians...

So for us to consider a sequence to be "random" it does not have to be *truly* random in any philosophical sense, but all that is really required is that without some inside knowledge, we are not able to predict the next value in the sequence from the preceding ones.

7.2 Uniform Distribution

We start by discussing how to sample from the uniform Unif$[0,1]$ distribution, then show how those may be used to sample from other distributions.

7.2.1 Linear Congruential Generators

One of the simplest types of PRNG, and the one originally used by Matlab, is called a *linear congruential generator* (LCG). Given parameters (a, m, c) and a starting value X_0, the generator uses the recurrence

$$X_{n+1} = (aX_n + c) \mod m,$$

where " $\mod m$" means taking the remainder after dividing by m. The inputs should satisfy $m > 0$, $0 < a < m$, $0 \leq c < m$, and $0 \leq X_0 < m$, and the subsequent outputs all satisfy $0 \leq X_n < m$. It follows that $U_n = X_n/m$ falls in the interval $[0, 1)$. When $c = 0$, the generator is sometimes called a *multiplicative congruential generator* or *Lehmer RNG*, after the aforementioned Lehmer.

■ **Example 7.1** Taking $a = 13$, $c = 0$, $m = 31$, and $X_0 = 1$ will produce a sequence beginning with

$$1, 13, 14, 27, 10, 6, 16, 22, 7, 29, 5, 3, \ldots$$

The first 30 terms will contain each integer from 1 to 30 exactly once, and the sequence then repeats itself. It has a *period* of $m - 1 = 30$. ■

Definition 7.2.1 The *period* of an LCG with seed X_0 is the smallest value $k > 0$ such that $X_k = X_0$. After this point, the LCG will indefinitely repeat itself in a pattern of length k.

So what choices of (a, c, m) make an LCG effective? We start with two considerations in particular:

- The LCG has a long period: it should be able to output many terms before repeating itself.
- Computing remainders modulo m should be inexpensive.

These two conditions do not guarantee that an LCG will be effective: for example, $a = 1$ and $c = 1$ will produce an LCG with period m, but it will not be a very good random number generator.

The second condition incentivizes making m either a power of 2 (in which case the remainder can be found simply by truncating the bits), or a closely related number such as $2^k \pm 1$. As for the first condition, if m is chosen to be a power of 2 then the maximal period is $m/4$, achieved if $a \equiv 3 \pmod 8$ or $a \equiv 5 \pmod 8$. If m is prime, then the maximal period is $m - 1$, achieved if a is a primitive root modulo m. As a result, Mersenne primes such as $2^{31} - 1$ or $2^{61} - 1$ are popular. In its earlier years, Matlab's `rand` function used the parameters

$$\begin{aligned} a &= 7^5 = 16807, \\ c &= 0, \\ m &= 2^{31} - 1 = 2147483647. \end{aligned}$$

The period is $m - 1$, or slightly larger than 2 billion. In the old days this was considered to be plenty, but it is not necessarily sufficient for large-scale modern applications.

One general extension of the LCG is the *multiple-recursive generator* (MRG), having the form

$$X_n = (a_1 X_{n-1} + a_2 X_{n-2} + \cdots + a_k X_{n-k}) \mod m,$$

for $k \geq 2$. If m is prime, the period of an MRG can be as large as $m^k - 1$.

7.2.2 Subtract With Borrow

Starting in 1994 with version 5, Matlab switched to using a PRNG developed by George Marsaglia known as *subtract-with-borrow*. It keeps 32 floating-point numbers between 0 and 1 in memory,

along with an integer index $0 \leq n \leq 31$, a random integer j for the seed, and a "borrow" flag b. At each step it uses the recurrence

$$U_n = U_{n+20} - U_{n+5} - b,$$

where the indices $(n+20)$ and $(n+5)$ are taken mod 32. If $U_n < 0$, then it is replaced with $U_n + 1$ and the flag b is set to 2^{-53} for the next step. If $U_n \geq 0$, then b is set to 0 zero instead. Taken as is, this method would have a period length of almost 2^{1430}.

One shortcoming is that not all floating-point numbers in $[0,1]$ are generated by this method as described, only the ones that are multiples of 2^{-53}. This covers all of the numbers in the interval $[1/2,1]$, but only half of those in $[1/4,1/2]$, one fourth of those in $[1/8,1/4]$, and so on. The random seed j is meant to fix this: the integer j is the result of a separate, independent random number generator, and where our output U_n has the floating point representation (s,e,f), the fractional bits f are XORed with j to produce the eventual output. With this modification, the PRNG has a period closer to 2^{1492}.

A more general relative of this setup is known as a *Lagged Fibonacci generator*, which uses the recurrence

$$X_n = X_{n-j} \star X_{n-k} \pmod{m}, 0 < j < k,$$

where $\star$ can be addition, subtraction, multiplication, or a bitwise XOR operation. The last k values must be kept in memory for this type of generator. The theory can get complex, but ideally an LFG is a relatively inexpensive method for extending the period of a PRNG.

7.2.3 Mersenne Twister

As of version 7.4 in 2007, Matlab's `rand` uses an algorithm known as the *Mersenne Twister*, developed in 1997 by Makoto Matsumoto and Takuji Nishimura. The version MT19937 has 32-bit outputs, stores 624 previous outputs in memory, and has a massive period of $2^{19937} - 1$. We won't go into the details, but it roughly uses the structure of the MRG as its foundation. The Matlab default version produces floating point values in the closed interval $[2^{-53}, 1 - 2^{-53}]$, so will never produce 0 or 1 exactly.

7.3 Non-Uniform Sampling

To sample from a non-uniform distribution, we assume that we have a reliable method for sampling from Unif$[0,1]$.

7.3.1 Finite Discrete Distributions

Suppose that the random variable X takes on vales $x_1 < x_2 < \cdots < x_n$ with probabilities $p_1, \ldots, p_n$. The CDF is then given by

$$F(x_i) = \sum_{j=1}^{i} p_j, \quad 1 \leq i \leq n.$$

Since $F(x_1) > 0$ and $F(x_n) = 1$. we can split the interval $[0,1]$ into the n disjoint intervals $[0, F(x_1)), [F(x_1), F(x_2)), \ldots, [F(x_{n-1}), F(X_n)]$. Given a random sample $U \sim$ Unif$[0,1]$, we can then set the value of X by determining which of these n intervals U falls within. This can be done efficiently by binary search.

■ **Example 7.2** For a Bernoulli Ber(p) distribution, we split the interval $[0,1]$ as $[0, 1-p) \cup [1-p, 1]$, where the first interval represents the outcome $X = 0$ and the second the outcome $X = 1$. We can therefore sample from a Bernoulli distribution with the Matlab command `rand >= (1-p)`. A simpler way to sample from the same distribution is `rand < p`. ■

7.3.2 Inverse Transform Sampling

If X is a continuous random variable whose CDF F_X is known, then we can sample X by properly transforming a variable U sampled from the uniform Unif$[0,1]$ distribution.

Theorem 7.3.1 — Universality of the uniform. Suppose that a random variable X has a continuous distribution with CDF F_X. Then the random variable U defined by $U = F_X(X)$ has a Unif$[0,1]$ distribution.

Proof. For any $u \in (0,1)$, we have

$$\begin{aligned} F_U(u) &= \mathbb{P}(U \le u) \\ &= \mathbb{P}(F_X(X) \le u) \\ &= \mathbb{P}(X \le F_X^{-1}(u)) \\ &= F_X(F_X^{-1}(u)) \\ &= u. \end{aligned}$$

■

The transformation used for inverse transform sampling is shown by a corollary of this theorem. As long as we can compute F_X^{-1} readily, we can sample from X by transforming U.

Corollary 7.3.2 Suppose that a random variable X has a continuous distribution with CDF F_X. If $U \sim$ Unif$[0,1]$, then $F_X^{-1}(U)$ has the same distribution as X.

■ **Example 7.3 — Uniform Sampling from a Circle.** Let $\Omega = \{(x,y) : x^2+y^2 = 1\}$ be a circle of radius 1. We would like to sample uniformly from Ω, meaning that the probability of an event E is proportional to the area of E. If we represent a random point as a Cartesian coordinate (X,Y), then X and Y are not independent. But in polar coordinates (R,Θ), we find that

$$\begin{aligned} F_{R\Theta}(a,b) &= \mathbb{P}(R \le a,\ \Theta \le b) \\ &= \frac{1}{\pi}\int_{r=0}^{a}\int_{\theta=0}^{b} r\,dr\,d\theta \\ &= \left(\frac{1}{\pi}\int_{r=0}^{a} 2\pi r\,dr\right)\left(\frac{1}{2\pi}\int_{\theta=0}^{b} d\theta\right) \\ &= \mathbb{P}(R \le a)\mathbb{P}(\Theta \le b). \end{aligned}$$

Thus R and Θ are independent, so to sample a random point in the circle we can sample R and Θ independently.

Sampling Θ is straightforward, since $\Theta \sim$ Unif$[0,2\pi]$. As for the radius, we get that

$$F_R(a) = \mathbb{P}(R \le a) = \frac{1}{\pi}\int_{r=0}^{a} 2\pi r\,dr = a^2,$$

so $F_R^{-1}(u) = \sqrt{u}$. We therefore sample $U \sim$ Unif$[0,1]$ and let $R = \sqrt{U}$. ■

Another example shows how we carried out the importance sampling strategy in Example 6.5.

■ **Example 7.4** A random variable X has the PDF $f_X(x) = \frac{4}{3}(1 - x/2)$ on the interval $[0,1]$. Its CDF is

$$F_X(x) = \frac{4}{3}\int_{t=0}^{a} 1 - \frac{x}{2}\,dx = -\frac{1}{3}a^2 + \frac{4}{3}a.$$

To find the inverse, we set up the system $-\frac{1}{3}a^2 + \frac{4}{3}a = u$, and using the quadratic formula gives the solutions

$$a = 2 \pm \sqrt{4 - 3U}.$$

Only the smaller root gives values in the interval $[0,1]$, so to sample X we sample $U \sim \text{Unif}[0,1]$ and return $X = F_X^{-1}(U) = 2 - \sqrt{4-3U}$. ∎

One final example gives us a method of taking two independent samples from a standard normal distribution using two independent $\text{Unif}[0,1]$ samples.

Theorem 7.3.3 — Box-Muller transform. Let $U_1, U_2 \overset{i.i.d.}{\sim} \text{Unif}[0,1]$. Then the random variables

$$Z_1 = R\cos\Theta = \sqrt{-2\ln U_1}\cos(2\pi U_2),$$
$$Z_2 = R\sin\Theta = \sqrt{-2\ln U_1}\sin(2\pi U_2)$$

are independent random variables with the standard normal $\text{Normal}(0,1)$ distribution.

Proof. Let $X, Y \overset{i.i.d.}{\sim} \text{Normal}(0,1)$, and let (R,Θ) be the corresponding polar coordinates. Then

$$\begin{aligned}
\mathbb{P}(R \le a,\ \Theta \le b) &= \int_{r=0}^{a}\int_{\theta=0}^{b} f_X(x) f_Y(y)\, r\, dr\, d\theta \\
&= \int_{r=0}^{a}\int_{\theta=0}^{b} \left(\frac{1}{\sqrt{2\pi}}e^{-x^2/2}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-x^2/2}\right) r\, dr\, d\theta \\
&= \frac{1}{2\pi}\int_{r=0}^{a}\int_{\theta=0}^{b} e^{-(x^2+y^2)/2}\, r\, dr\, d\theta \\
&= \left(\int_{r=0}^{a} re^{-r^2/2}\, dr\right)\left(\frac{1}{2\pi}\int_{\theta=0}^{b} d\theta\right) \\
&= \mathbb{P}(R \le a)\mathbb{P}(\Theta \le b).
\end{aligned}$$

As was the case in Example 7.3, R and Θ are independent with $\Theta \sim \text{Unif}[0, 2\pi]$. We therefore set $\Theta = 2\pi U_2$.

As for R, we get that the CDF is

$$F_R(a) = \int_{r=0}^{a} re^{-r^2/2}\, dr = 1 - e^{-a^2/2},$$

and solving $1 - e^{-a^2/2} = u$ shows that the inverse is given by

$$F_R^{-1}(u) = a = \sqrt{-2\ln(1-u)}.$$

But if U is a uniform $\text{Unif}[0,1]$ random variable then so is $1-U$, so we can simplify the equation to $R = \sqrt{-2\ln U_1}$. ∎

7.3.3 Rejection Sampling

The basic idea of rejection sampling is as follows: suppose that a random variable X has pdf f_X satisfying

$$\max_{x\in\mathbb{R}} f_X(x) = M,$$
$$f_X(x) = 0, \quad x \notin [a,b],$$

for some finite numbers $M, a, b \in \mathbb{R}$. Then we can encase the graph of f_X within the rectangle $[a,b] \times [0,M]$. If we throw darts randomly in this rectangle and retain only the ones under the curve, these remaining darts will be uniformly distributed over the area under the curve. Furthermore, their x-coordinates will have the density f_X. Intuitively, this is because there is more room for the darts to land where f_X is greatest.

Now, suppose we only generate a random x-coordinate in $[a,b]$. Looking at generating the y-coordinate in $[0,M]$, we observe that the probability of retaining the sample is equal to $f(x)/M$. This observation lends itself to the following approach:

Theorem 7.3.4 — Rejection Sampling. Let X be a random variable whose PDF f_X is uniformly bounded by M and is zero outside of the interval $[a,b]$. Sample x according to the algorithm

1. Sample y from $\text{Unif}[a,b]$ and $u \sim \text{Unif}[0,1]$;
2. If $u < f(y)/M$, return $x = y$. Otherwise, return to step 1.

Then x will be a sample from distribution X, and the algorithm will take an average of M iterations to obtain a sample.

Many probability distributions of interest are nonzero over an infinite interval, of course. To accomodate such cases, the previous theorem can be generalized in the following manner.

Theorem 7.3.5 — Generalized Rejection Sampling. Let X be a random variable with PDF f, and let Y be a random variable whose pdf g satisfies $g(x) > 0$ whenever $f(x) > 0$, and $f(x)/g(x) \leq M$. Sample x according to the algorithm

1. Sample y from Y and $u \sim \text{Unif}[0,1]$;
2. If $u < f(y)/(Mg(y))$, return $x = y$. Otherwise, return to step 1.

Then x will be a sample from distribution X, and the algorithm will take an average of M iterations to obtain a sample.

The "ideal" distribution g in terms of minimizing M is $g(x) = f(x)$, but this is impractical since sampling Y will then be just as difficult as sampling X. In practice, it has to be significantly simpler to sample from g, and the maximum ratio M should not be too large.

■ **Example 7.5** Suppose we want to sample from a distribution X with PDF $f_X(x) = \frac{4}{3}(1 - x/2)$ on the interval $[0,1]$. The maximum height of this function is $M = 4/3$. We can sample a point x uniformly from $[0,1]$, then accept it with probability $f_X(x)/M = 1 - x/2$. On average 75 percent of points will be accepted, and as opposed to the transformation method of Example 7.4 we do not need to compute any square roots. ■

■ **Example 7.6 — Uniform Sampling From a Circle.** To sample uniformly from a unit circle, sample x and y independently from a $\text{Unif}[-1,1]$ distribution. If $x^2 + y^2 \leq 1$, keep the sample. Otherwise, try again.

The disadvantage of this approach over the one in Example 7.3 is that we will need an average of $(4/\pi) \cdot 2 \approx 2.55$ samples rather than just 2, since the probability of acceptance is only $\pi/4$. The advantage is that we do not have to compute a square root, which can be more expensive than the multiplication needed to check whether the condition $x^2 + y^2 \leq 1$ is satisfied.

Formally, to show how this is an example of the general method described, we define the probability distribution $f_{XY} = \frac{1}{\pi} \mathbb{1}_{x^2+y^2 \leq 1}$ over two dimensions. This joint PDF is enclosed in a cube $[-1,1] \times [-1,1] \times [0,1/\pi]$, and we keep points falling under the surface defined by f_{XY}. ■

As for normal distributions, we can improve upon the Box-Muller transform by incorporating rejection sampling.

Theorem 7.3.6 — Marsaglia polar method. Sample random points (x,y) independently from a $\text{Unif}[-1,1]$ distribution until the condition $s = x^2 + y^2 \leq 1$ is satisfied. Then the random variables

$$Z_1 = R\cos\Theta = \sqrt{-2\ln s}\cdot\frac{x}{\sqrt{s}},$$
$$Z_2 = R\sin\Theta = \sqrt{-2\ln s}\cdot\frac{y}{\sqrt{s}}$$

are independent random variables with the standard normal $\text{Normal}(0,1)$ distribution.

Proof. Upon acceptance, the point (x,y) will come from a uniform distribution over the circle, whose polar coordinates $(\sqrt{s},\theta)$ are independent. It was established in Example 7.3 that the CDF of the radius is $F(a) = a^2$, so by Theorem 7.3.1 it follows that $F(\sqrt{s}) = s$ has a uniform distribution. We can then apply the Box-Muller transform with $U_1 = s$, $\cos(2\pi U_2) = \cos(\theta) = x/\sqrt{s}$, and $\sin(2\pi U_2) = \sin(\theta) = y/\sqrt{s}$. ■

The Marsaglia polar method will have to take more samples on average than the regular Box-Muller transform by a factor of $4/\pi \approx 1.27$. In exchange, it avoids the more expensive trigonometric operations in favor of computing $s = x^2 + y^2$ and $(x/\sqrt{s}, y/\sqrt{s})$. On the other hand, it has been suggested that on more modern machines the evaluation of the conditional `if` statement might be costly enough to eliminate whatever advantage was otherwise gained.

7.3.4 Ziggurat Algorithm

If the principal idea behind rejection sampling can be summarized as "enclose the graph of the PDF in a large rectangle", then the idea behind the Ziggurat algorithm can be similarly summarized as "enclose the graph of the PDF in a collection of rectangles of equal area". The goal is that using more rectangles means that we can better approximate the graph of the PDF and therefore reject fewer samples, while still being able to sample cheaply from the enclosing distribution.

The key idea is that if the enclosing shape is divided into parts of equal area, then the procedure "choose a part uniformly at random, then sample uniformly from that part" is equivalent to "sample uniformly from the whole shape". Figure 7.1 illustrates an example where the normal distribution is split into 8 levels. If the number of levels n is a power of 2 (say, $n = 2^k$), then we can cheaply sample an index $0 \leq i < n = 2^k$ by sampling k random bits. After that, a point within the rectangle can be chosen from a uniform distribution by sampling two uniform variables $X, Y \sim \text{Unif}[0,1]$. Common choices include $n = 128$ or $n = 256$, and in the former case we only need to resample a little less 3 percent of the time.

The area and locations of the rectangles are not trivial to compute, but once a solution is found it can be saved and reused for any future random number generation. This is how Matlab's current version of `randn` works in practice, using a precomputed lookup table for the coordinates of the rectangles. Some extra work must also be done if the lowest level is selected, in order to account for the infinite tail of the distribution.

7.4 Randomness Testing

Random number generators can vary widely in quality, so how do we tell in practice if one is good or bad? From one point of view, this question might appear a bit strange: if we have a process generating random bits, then any two given sequences of bits are equally likely to occur. Can we really say that one is more "random" than another? Furthermore, we know that the PRNGs we have discussed are deterministic, so none of them are really random at all!

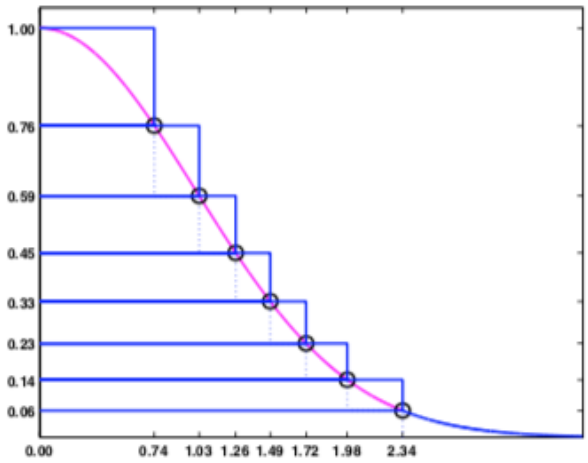


Figure 7.1: Ziggurat method with 8 levels on the positive half of a normal distribution. Source: [17].

Figure 7.2: Dilbert ©2001, Scott Adams. Image accessed from [9].

■ **Example 7.7** Generate 10 bits at random. The sequences 0000000000 and 1001100010 both have a $2^{-10} = 1/512$ chance of occurring. ■

However, there are a few standard tests that a randomly generated sequence of number should satisfy with high probability. If a block of output fails one of these tests it does not necessarily mean that the RNG is poor – all of the tests should be failed with a certain probability, after all – but if an RNG fails many of these tests consistently then we would have good cause to be suspicious.

R A good random number generator will occasionally produce sequences that look "non-random" to the human eye and which fail standard statistical tests for randomness!

7.4.1 Histogram Test

The first test is simple: if a PRNG is supposed to produce numbers from a Unif$[0,1]$ distribution, then a histogram of sampled values should have roughly the same number of samples in each bin.

As an example, we'll compare three 0/1 bit sequences produced by different methods. The first uses Matlab's `rand`, generating N bits with the code `rand(1,N)<0.5`. The second sequence was produced by students from a previous semester, who were instructed to come up on their own with bit sequences that "looked random". The third was produced with the PRNG used in the game Dota 2 [5], which was deliberately designed to prevent long streaks of successes or failures for events

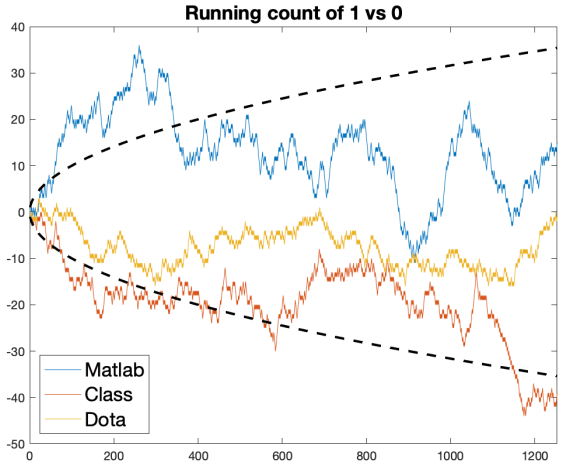


Figure 7.3: Running counts for bit sequences.

with random outcomes and thus limit the extent to which good or bad luck can affect a player's fortunes.

The first 64 bits from each sequence are shown below. Can you tell which is which just by looking?

```
1100100110011001011011110110100101110001111011001000111011111011
0010011001101000111010101101000101010000000110110100101011111100
0011010010101010111110100101001010011001010010101001011101101011
```

Figure 7.3 shows a plot for the running count of the number of 1's (positive) versus zeros (negative) for each method. The black lines at $\pm\sqrt{n}$ give an approximate 95 percent confidence interval for the running count at any given value of n. Given the knowledge that the Dota 2 PRNG is designed to prevent streaks, we might be able to guess which line in the plot represents that method. The Matlab sequence and student-generated sequence, however, don't look especially different.

For the histogram test, we bin the bits in groups of 4 and interpret each as a number from 0 to 15. The histogram test, presented in Figure 7.4 shows that all three sequences have very different behavior. The Matlab histogram counts are fairly evenly distributed (at least, as far as we can tell given the smallish sample size of $N \approx 1250$). We shouldn't expect the histogram to be *exactly* evenly distributed due to random variation, but we could more formally use a goodness-of-fit test such as Pearson's chi-squared test to decide whether

The student generated sequences disproportionately favor the bins $5 = \texttt{0101}$ and $10 = \texttt{1010}$, indicating that the sequences that the students thought "looked random" flipped between 0 and 1 far more often than they should have. Conversely, streaks were comparatively rare: the least common bins were $0 = \texttt{0000}$ and $12 = \texttt{1100}$. The behavior for the Dota 2 sequence is even more striking: by design, the string `0000` *never* appeared! So both the human-generated sequence and the Dota 2 sequence exhibit some amount of predictable behavior, and indicate that the underlying methods are not suitable as general-purpose PRNGs.

The histogram test is a simple one, but for some applications it can be sufficient. With the Monte Carlo methods discussed in chapter 6, we don't necessarily need the samples to be truly independent as long as the batch of samples $\{x_1, \ldots, x_n\}$ *in aggregate* has the distribution that we might expect of independent samples. The distinction can be useful when we want to sample from

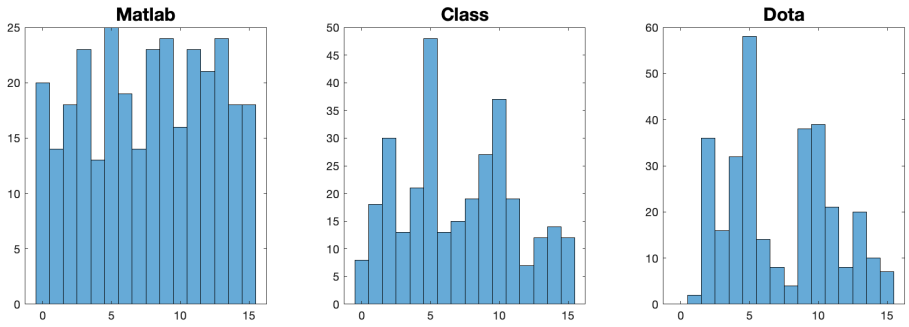


Figure 7.4: Histogram test for bit sequences from three different sources.

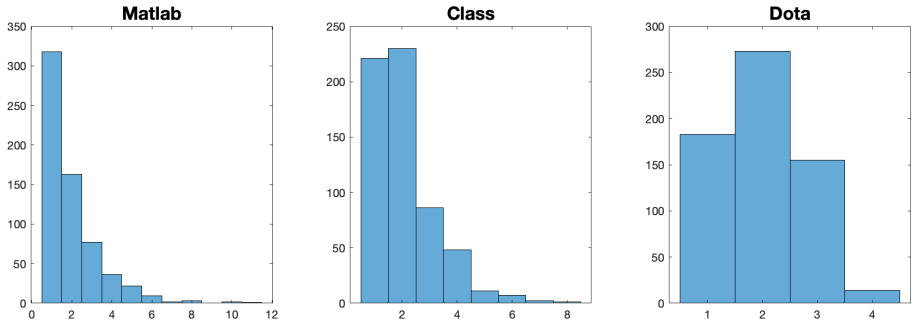


Figure 7.5: Run length test for bit sequences from three different sources.

a distribution f_X that is complicated enough to make the sampling strategies discussed in Section 7.3 impractical. Certain methods such as the Metropolis-Hastings algorithm can save time by producing samples that are not necessarily independent, but which in aggregate produce histograms that resemble that resemble the desired distribution.

7.4.2 Run Lengths

A slightly more sophisticated test involves examining the run lengths: once we have encountered a 1 in a given bit sequence, how many more bits later will we encounter the next 1? If the bits are all independent Bernoulli $\text{Ber}(p)$ variables, then the distance between successive zeros will have a geometric $\text{Geo}(p)$ distribution.

Figure 7.5 shows the histograms of run lengths for our three example bit sequences. Once again, the differences between the three are clear: the Matlab-generated sequence has an approximate $\text{Geo}(0.5)$ distribution, with each run length appearing about half as often as the previous one. The student-generated sequence massively underrepresents the sequence `11` compared to the sequence `101` (the former ought to appear about twice as often), as does the Dota 2 sequence. In the Dota 2 sequence, furthermore, the sequence `10001` almost never appears, and the sequence `0000` doesn't appear at all.

Interestingly, the run length test exposes a defect in Matlab's legacy subtract-with-borrow PRNG (Section 7.2.2). The code in Program 7.1 generates 2^{24} random samples from the $\text{Unif}[0,1]$ distribution and counts the run lengths where a "success" is registered when $x_i < \frac{1}{100}$. The run lengths should follow the geometric distribution $\text{Geo}(\frac{1}{100})$.

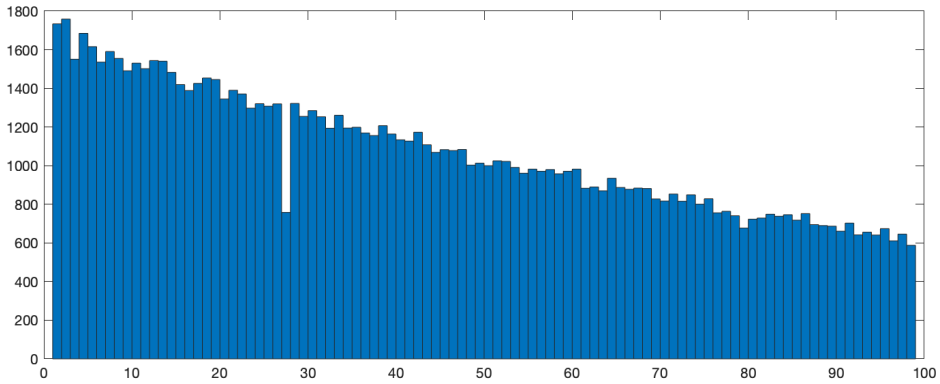


Figure 7.6: Run length test for Matlab's legacy generator.

Program 7.1: Run length test for Matlab's legacy generator. Source: [17]

```
rand('state',0)
x = rand(1,2^24);
thresh = 0.01;
k = diff(find(x<thresh));
t = 1:100;
c = histcounts(k,t);
b = bar(t(1:end-2),c(1:end-1),'histc');
```

Results are shown in Figure 7.6 for runs up to length 99. The distribution looks mostly as we would expect it to, except for one massive outlier where runs of length 27 occur far less often than they ought to. As it turns out, this is no coincidence. In the recurrence $U_n = U_{n+20} - U_{n+5} - b$ used for that PRNG, the indices are taken mod 32, and the odd behavior is related to the fact that $32 - 5 = 27$.

7.4.3 Spectral Test

One simple but powerful test specific to linear congruential generators (Section 7.2.1) is known as the spectral test. The idea is that because these tests use a one-term recurrence $X_{n+1} = (aX_n + c) \bmod m$, the pairs (X_n, X_{n+1}) when plotted in 2 dimensions will fall on a finite number of lines. More generally, the plots of k-tuples of successive outputs $(X_n, X_{n+1}, \ldots, X_{n+k-1})$ in k dimensions will fall on a finite number of hyperplanes. Marsaglia [15] provides a proof for the case $X_{n+1} = aX_n \bmod m$ (i.e., $c = 0$), and notes that a similar result holds when $c \neq 0$.

Theorem 7.4.1 If $c_1, c_2, \ldots, c_n$ is any choice of integers such that

$$c_1 + c_2 a + c_3 a^2 + \ldots + c_k a^{k-1} \equiv 0 \pmod{m},$$

then the point $(U_n, U_{n+1}, \ldots, U_{n+k-1}) = (X_n/m, X_{n+1}/m, \ldots, X_{n+k-1}/m)$ must lie in one of the planes

$$c_1 x_1 + c_2 x_2 + \ldots + c_k x_k = j, \quad j \in \mathbb{Z}$$

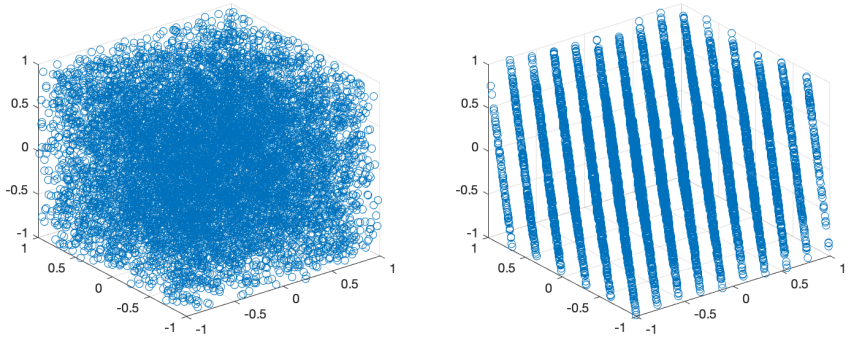


Figure 7.7: Left: Matlab's legacy `rand` function. Right: IBM's `RANDU` generator.

and the number of such planes that intersect the (open) unit hypercube $(0,1)^k$ is at most

$$|c_1| + |c_2| + \ldots + |c_k| - 1.$$

Proof. From the LCG recurrence it can be checked that $X_{n+i} \equiv a^i X_n \pmod{m}$ in general. Thus

$$c_1 X_n + c_2 X_{n+1} + \ldots + c_k X_{n+k-1} \equiv X_n(c_1 + c_2 a + c_3 a^2 + \ldots + c_k a^{k-1}) \equiv 0 \pmod{m}.$$

Since $U_{n+i} = X_{n+i}/m$, it follows that $c_1 U_n + c_2 U_{n+1} + \ldots + c_k U_{n+k-1}$ is always an integer. As for the bound on the number of hyperplanes, we find that for $\mathbf{x} \in (0,1)^n$ the bounds

$$\sum_{i=1}^{n} \min\{c_i, 0\} \cdot 1 < \sum_{i=1}^{n} c_i x_i < \sum_{i=1}^{n} \max\{c_i, 0\} \cdot 1$$

will hold. The number of integers that fall between the two bounds is strictly less than the difference between the upper and lower bounds, which is $|c_1| + |c_2| + \ldots + |c_n|$. ■

An infamous example of a PRNG failing the spectral test is the `RANDU` generator used by IBM, an LCG with parameters $a = 65539$, $c = 0$, and $m = 2^{31}$. It can be shown that this choice of parameters satisfies

$$9x_k - 6x_{k+1} + x_{k+2} \equiv 0 \pmod{m}$$

for all k, so by Theorem 7.4.1 all consecutive triplets must lie within one of at most 15 planes. Figure 7.7 demonstrates the deficiency in comparison with a previous implementation of Matlab's `rand`, an LCG using $a = 16807$, $c = 0$, and $m = 2^{31} - 1$. Interestingly, despite its massive shortcomings as a random number generator `RANDU` can still accurately estimate the volume of a sphere using a 3-D version of the Monte Carlo sampling strategy in Section 6.1.1.

7.4.4 Other Tests

Here we give a small sample of other methods for determining the quality of a random number generator. One early collection is known as the *Diehard* test suite, developed by Marsaglia. Among its tests are the following:

- Birthday spacings: choose a large number random points from a fixed interval. The spacings between the points should asymptotically follow an exponential distribution.
- Matrix ranks: Generate matrices with random $\{0,1\}$ entries and certain dimensions (32×32, 31×31, or 6×8). The ranks of these matrices should follow a certain known distribution.

- Permutations: Look at sequences of five consecutive random numbers, and note what order they appear in by magnitude. The $5! = 120$ possible orderings should appear with equal probability.
- Minimum distance: Place 8000 points in a square, then find the minimum distance between the points. The square of this distance will follow a certain exponential distribution.

A more recent collection is the NIST test suite [1]. It has some tests in common with the Diehard suite, but also includes the following (among others):

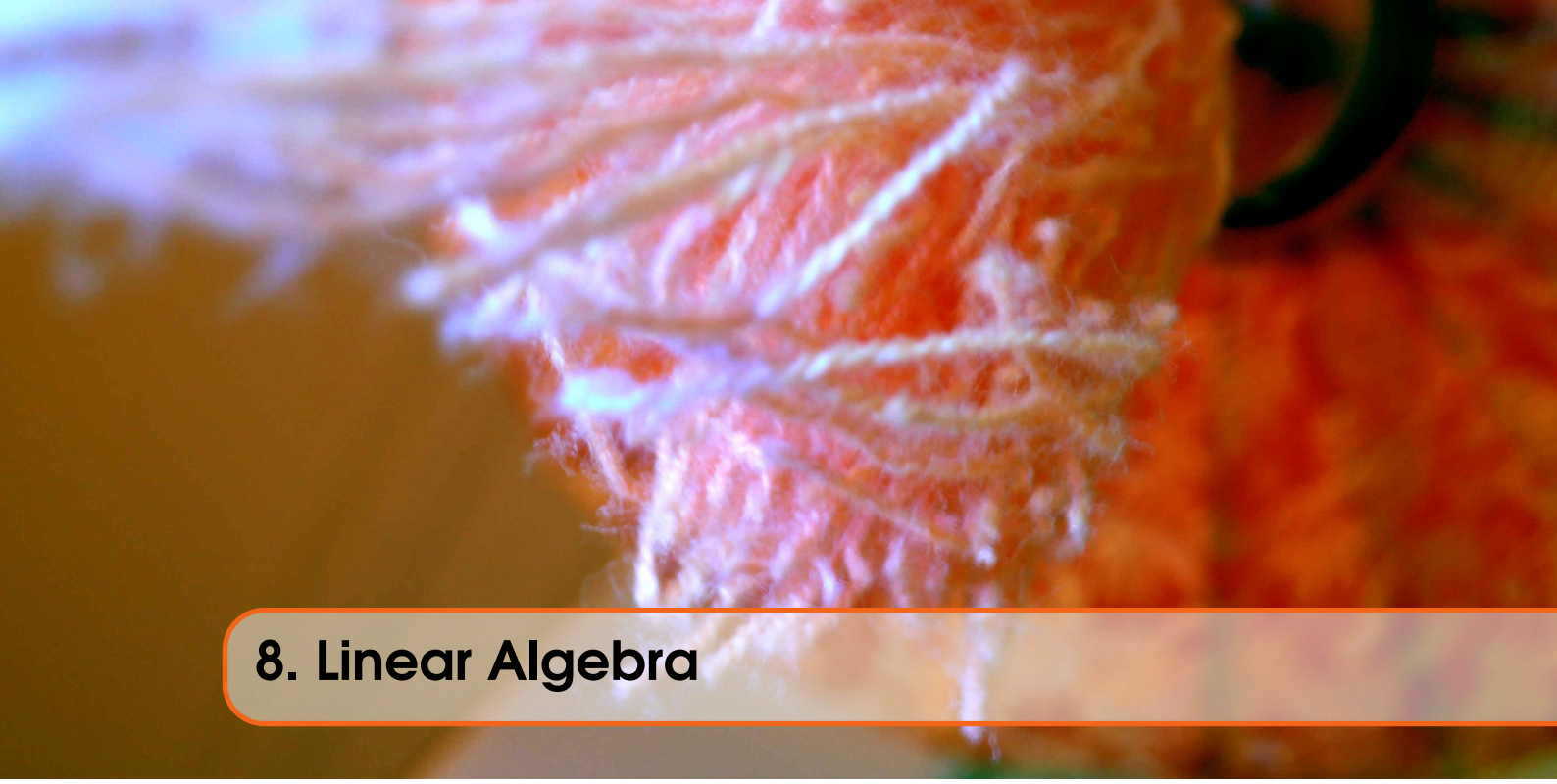
- Random Excursions Test: View a sequence of $\{0,1\}$ bits as a random walk in 1 dimension, as is done in Figure 7.3. For any k, the number of times the running sum is equal to k follows a certain known distribution. With $k = 0$ in particular, the number of times a true random walk is expected to cross the zero line grows roughly proportional to $\sqrt{n}$.

7.5 Matlab Commands

- `RandStream`: Create a stream of uniform pseudorandom numbers. By default, the functions `rand`, `randi`, `randn` all draw from the single stream called the *global random number stream*. Using `RandStream`, you can set up multiple different streams simultaneously, even using different random number generation algorithms.
- `RandStream.list`: Lists available random number generator algorithms.

III

Part Three



8. Linear Algebra

Now we take a more focused look at the basic tools of linear algebra. When dealing with problems at large scale, we should always be mindful of the *cost* of basic operations, either in terms of the time (or by proxy, the number of arithmetic operations required) or the memory needed.

One simple model for determining the cost of an algorithm is to count the number of flops it uses, but this is a bit of an oversimplification. On modern computers, arithmetic is cheap and moving data around is expensive, not to mention that many computers have multiple cores that can carry out instructions in parallel. Nonetheless, flop counts can give us at least a rough idea of how long an computer might require to carry out a given algorithm.

R Some linear algebra subroutines are common enough that computer manufactures have standardized them as *Basic Linear Algebra Subroutines* (BLAS) and optimized them for their machines [4]. These subroutines are available through C or Fortran interfaces, but under the hood they have been optimized for the specific machine. As of the year 2000 MATLAB has used LAPACK as its foundation, a sofware package written in Fortran 90 which in turn is designed to use BLAS as much as possible [13, 14].

8.1 Cost of Basic Operations

8.1.1 Vector Operations

The following operations all require at most $\mathcal{O}(n)$ flops and $\mathcal{O}(n)$ data. For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, and a scalar $\alpha \in \mathbb{R}$,

- Computing $\alpha\mathbf{x}$ requires n multiplies.
- Computing $\mathbf{x}+\mathbf{y}$ requires n adds.
- Computing the dot product $\mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i$ requires n multiplies and $n-1$ adds.
- Computing $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T\mathbf{x}}$ requires n multiplies and $n-1$ adds, plus a square root.
- Computing $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$ requires $n-1$ adds. The bitwise operation to take the absolute values is relatively cheap compared to the cost of the adds.

8.1.2 Matrix-Vector Operations

The next set of operations require $\mathcal{O}(mn)$ flops and data for a rectangular matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, or $\mathcal{O}(n^2)$ flops and data for a square matrix.

- For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and vector $\mathbf{x} \in \mathbb{R}^n$, computing $\mathbf{Ax}$ requires mn multiplies and $m(n-1)$ adds.
- If $\mathbf{A} \in \mathbb{R}^{n\times n}$ is triangular (Definition 8.1.1) and $\mathbf{b} \in \mathbb{R}^n$, solving $\mathbf{Ax} = \mathbf{b}$ requires n^2 flops.

Definition 8.1.1 An *upper triangular matrix* is one whose entries below the main diagonal are uniformly zero:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots \\ 0 & & & a_{nn} \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

A *lower triangular* matrix is defined similarly.

8.1.3 Matrix-Matrix Product

For square matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$, computing the product requires $\mathcal{O}(n^3)$ flops. This is costly enough that a personal computer may take some time when the matrix sizes are in the thousands.

- For matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$, computing $\mathbf{AB}$ requires mnp multiplies and $m(n-1)p$ adds.

Definition 8.1.2 For matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_p \end{bmatrix} \in \mathbb{R}^{n\times p}$, the matrix product $\mathbf{AB}$ can be defined columnwise as

$$\mathbf{AB} = \begin{bmatrix} \mathbf{Ab}_1 & \cdots & \mathbf{Ab}_p \end{bmatrix} \in \mathbb{R}^{m\times p}.$$

It requires mnp multiplies and $m(n-1)p$ adds.

■ **Example 8.1** The order of computation can have a large effect on the cost. For $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, and $\mathbf{x} \in \mathbb{R}^p$, computing $\mathbf{A}(\mathbf{Bx})$ requires about $2(m+p)n$ flops, but computing $(\mathbf{AB})\mathbf{x}$ requires about $2mnp$ flops! ■

8.1.4 Solving Linear Systems

The most straightforward way to solve the system $\mathbf{Ax} = \mathbf{b}$ is to use Gaussian elimination. For $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$,

- Solving $\mathbf{Ax} = \mathbf{b}$ through Gaussian elimination costs about $\frac{2}{3}n^3$ flops.
- If $\mathbf{A}$ is symmetric, the above cost is reduced to about $\frac{1}{3}n^3$.
- Forming $\mathbf{A}^{-1}$, which is equivalent to solving $\mathbf{AX} = \mathbf{I}_n$, requires about $2n^3$ flops.

Solving a linear system $\mathbf{Ax} = \mathbf{b}$ does not require computing $\mathbf{A}^{-1}$ explicitly! In MATLAB, it's faster to use `A\b` rather than `inv(A)*b`.

8.1.5 Special Matrices

For problems with special structure, some of the costs listed in this section may be considerably reduced.

Zero matrix

The zero matrix $\mathbf{0}_{m\times n}$ satisfies $\mathbf{0}_{m\times n}\mathbf{x} = \mathbf{0}_m$ for all $\mathbf{x} \in \mathbb{R}^n$. This operation requires no flops. The zero matrix is not invertible.

Identity matrix

The identity matrix satisfies $\mathbf{I}_n\mathbf{x} = \mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^n$. This "matrix-vector product" requires no flops at all. Neither does solving $\mathbf{I}_n\mathbf{x} = \mathbf{b}$, as the solution is simply $\mathbf{x} = \mathbf{b}$.

Diagonal matrices

A diagonal matrix is a matrix whose nonzero entries lie entirely on the main diagonal:

$$\mathbf{D} = \begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{nn} \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

In MATLAB, the `diag` operator takes a matrix and returns a vector with the entries on its main diagonal. Conversely, it also takes a vector and returns a diagonal matrix whose diagonal entries are the elements of the vector. Thus, if $\mathbf{d} = \begin{bmatrix} d_{11} & d_{22} & \cdots & d_{nn} \end{bmatrix}^T \in \mathbb{R}^n$, then $\mathbf{D} = \texttt{diag}(\mathbf{d})$ and $\mathbf{d} = \texttt{diag}(\mathbf{D})$.

- Matrix-vector multiplication with a diagonal matrix requires only n flops:

$$\mathbf{D}\mathbf{x} = \begin{bmatrix} d_{11}x_1 \\ \vdots \\ d_{nn}x_n \end{bmatrix}.$$

- Matrix-matrix multiplication requires n^2 flops if one matrix is diagonal, or n flops if both are diagonal. Multiplication from the left by a diagonal matrix has the effect of scaling the rows, and multiplication from the right has the effect of scaling the columns:

$$\mathbf{D}\mathbf{A} = \begin{bmatrix} d_{11}a_{11} & d_{11}a_{12} & \cdots & d_{11}a_{1n} \\ d_{22}a_{21} & d_{22}a_{22} & & \\ \vdots & & \ddots & \\ d_{nn}a_{n1} & & & d_{nn}a_{nn} \end{bmatrix}, \qquad \mathbf{A}\mathbf{D} = \begin{bmatrix} a_{11}d_{11} & a_{12}d_{22} & \cdots & a_{1n}d_{nn} \\ a_{21}d_{11} & d_{22}a_{22} & & \\ \vdots & & \ddots & \\ a_{n1}d_{11} & & & a_{nn}d_{nn} \end{bmatrix}.$$

- A diagonal matrix is invertible if and only if all of its diagonal elements are nonzero. Computing the inverse requires n flops, as does solving linear systems:

$$\mathbf{D}^{-1} = \begin{bmatrix} 1/d_{11} & & & \\ & 1/d_{22} & & \\ & & \ddots & \\ & & & 1/d_{nn} \end{bmatrix}, \qquad \mathbf{D}^{-1}\mathbf{b} = \begin{bmatrix} b_1/d_{11} \\ b_2/d_{22} \\ \vdots \\ b_n/d_{nn} \end{bmatrix}.$$

Row and column selection

Multiplication of a matrix with a standard basis vector (Definition 3.1.1) has the effect of selecting a single row or column from the matrix, requiring data movement but no arithmetic:

$$\mathbf{A}\mathbf{e}_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{nj} \end{bmatrix} \qquad \mathbf{e}_j^T\mathbf{A} = \begin{bmatrix} a_{j1} & a_{j2} & \cdots & a_{jn} \end{bmatrix}.$$

Symmetric matrices

When a matrix $\mathbf{A}$ is symmetric, certain problems (such as solving systems of equations or finding eigenvalues) can be solved more efficiently.

- If $\mathbf{A} \in \mathbb{R}^{n\times n}$ is symmetric, solving $\mathbf{Ax} = \mathbf{b}$ through (a modified version of) Gaussian elimination costs about $\frac{1}{3}n^3$ flops.

This saves a factor of 2 over the nonsymmetric case. When $\mathbf{A}$ is not just symmetric but also has all positive eigenvalues (in which case we say $\mathbf{A}$ is *positive definite*), it has a special type of factorization.

Definition 8.1.3 For a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with all positive eigenvalues, the *Cholesky factorization* is a factorization

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T,$$

where $\mathbf{L}$ is lower triangular. Equivalently, we may write it as $\mathbf{A} = \mathbf{R}^T\mathbf{R}$, where $\mathbf{R} = \mathbf{L}^T$ is upper triangular.

Computing the Cholesky factorization requires about $\frac{1}{3}n^3$ flops.

8.2 Matrix Applications

What is a matrix, anyway, and what is it good for? There are a few potential ways to view a matrix, and which one is most useful depends on the application.

8.2.1 Table Interpretation

The most straighforward way to interpret a matrix is as a box of numbers, indexed by a row and column. One example of this is image data, where an $m \times n$ grayscale image can be stored as an $m \times n$ matrix. The following three image formats are all supported by Matlab:

- A grayscale image, stored as an $m \times n$ matrix of values in $[0, 1]$ (for single or double precision) or $[\texttt{intmin}, \texttt{intmax}]$ (for integer formats).
- An RGB image, stored as an $m \times n \times 3$ array of R-G-B values. (you could think of this as three $m \times n$ matrices stacked on top of one another). Matlab's `rgb2gray` function uses the weighted sum `X = 0.2989*R + 0.5870*G + 0.1140*B`.
- An *indexed image* has two components: a matrix $X \in \mathbb{R}^{m\times n}$ whose entries are integers between 1 and p, and a map $M \in \mathbb{R}^{p\times 3}$ containing p different RGB values. This format can be useful when storing an image with few distinct colors (small p), because X will require only a single integer entry instead of one for each of the RGB values.

Another common format is sometimes called an *object-attribute* matrix. The rows represent individuals (objects), and the columns represent measurements made on those individuals (attributes).

- A dataset on joke ratings where m people each rated n jokes could be stored as an $m \times n$ matrix, where the (i, j) entry contains the rating person i gave to joke j.
- The *Netflix Prize* dataset contained over 100 million ratings that 480,189 users gave to 17,770 movies. These ratings could be interpreted as entries of a 480189×17770 matrix, but most of the data in the matrix is *missing* as most users have not rated most movies.
- Performance results from a decathlon could be stored as an $n \times 10$ matrix, where the (i, j) entry contains the result athlete i got in event j.

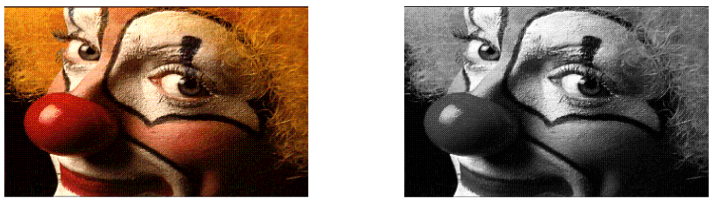


Figure 8.1: Grayscale and color representation's of Matlab's `clown` data.

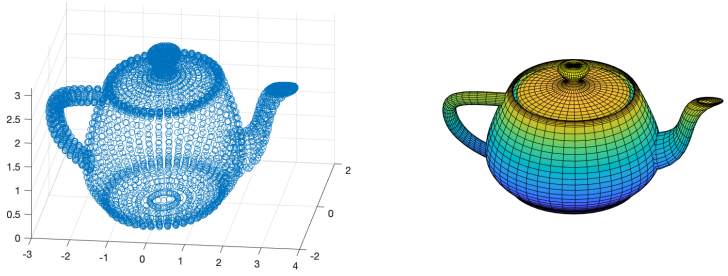


Figure 8.2: Vertices and faces of the Utah teapot.

8.2.2 Collection of Vectors

Another potential interpretation is that a matrix is just a handy way to store a collection of vectors. The *object-attribute* matrices could be interpreted in this manner: for example, with the dataset on joke ratings, a single column vector represents all of the ratings given to a single joke. Similarly, a single row vector contains all the ratings given by a single person.

More directly, this format for matrices has direct applications in computer graphics, where an object can be represented by a collection of vertices and faces in $\mathbb{R}^3$. The *Utah teapot*, for example, can be accessed in Matlab as the variable `teapotGeometry`. It has 4806 vertices in $\mathbb{R}^3$, which can be represented in the form of a 4608 matrix.

8.2.3 Linear Transformation

A substantially different way to view a matrix is as a *linear transformation*: an $m \times n$ matrix represents a *function* that takes a vector in $\mathbb{R}^n$ and transforms it into a vector in $\mathbb{R}^m$. The fact that it is *linear* means that it is additive ($\mathbf{A}(\alpha\mathbf{x}+\mathbf{y}) = \alpha\mathbf{A}\mathbf{x}+\mathbf{A}\mathbf{y}$), and more generally that it maps subspaces to subspaces.

- Figure 8.3 demonstrates two linear transformations. Taking the point cloud and two matrices

$$\mathbf{R} = \begin{bmatrix} -\frac{1}{4} & -\frac{1}{4} & 0 & \frac{1}{4} & 0 & -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & -\frac{1}{2} \end{bmatrix}, \qquad \mathbf{A}_1 = \begin{bmatrix} -\frac{3}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \qquad \mathbf{A}_2 = \begin{bmatrix} \frac{1}{5} & \frac{4}{5} \\ -\frac{2}{5} & \frac{2}{5} \end{bmatrix},$$

  the matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ can be taken as functions from $\mathbb{R}^2$ to $\mathbb{R}^2$. Figure 8.3 demonstrates the effect of these transformations, plotting the shapes described by $\mathbf{R}$, $\mathbf{A}_1\mathbf{R}$, and $\mathbf{A}_2\mathbf{R}$.
- The sum and average of a collection of elements in a vector $\mathbf{a} \in \mathbb{R}^n$ can be found by using the all-ones vector:

$$\sum_{i=1}^{n} a_i = \mathbb{1}^T\mathbf{a}, \qquad \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{1}{n}\mathbb{1}^T\mathbf{a}.$$

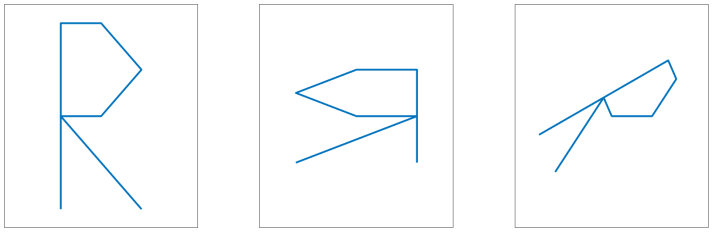


Figure 8.3: Plot of the letter "R" and two linear transformations of its vertices.

- If a degree-2 polynomial $a_2x^2 + a_1x + a_0$ is represented by a vector $\mathbf{a} \in \mathbb{R}^3$, then polynomial multiplication by a degree-3 polynomial given by $\mathbf{b} \in \mathbb{R}^4$ can be represented as the matrix-vector product

$$\mathbf{b} * \mathbf{a} = \mathbf{a} * \mathbf{b} = \begin{bmatrix} b_3 & 0 & 0 \\ b_2 & b_3 & 0 \\ b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 \\ 0 & b_0 & b_1 \\ 0 & 0 & b_0 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} \in \mathbb{R}^6.$$

  Matrices with the sort of structure as the one above (constant along the diagonals) are known as *Toeplitz* matrices.

8.2.4 Representing Graphs

A graph can be represented by a set of n *vertices* $\{1, \ldots, n\}$, and *edges*, represented as pairs (i, j) where $1 \leq i \leq n$ and $1 \leq j \leq n$. Graphs can be *directed* if the pairs (i, j) are ordered pairs, or *undirected* if the pairs are unordered. Figure 8.4 shows an example for an directed graph. Graphs are commonly represented by $n \times n$ *adjacency matrices*, which have a nonzero entry at index (i, j) if there is an edge from i to j (or vice versa, depending on convention). Adjacency matrices for undirected graphs will be symmetric, with a nonzero entry at both (i, j) and (j, i).

A second common form of representation is an $n \times m$ *incidence matrix*. In this case, each column k represents an edge and has two nonzero entries: if there is an edge from vertex i to vertex j, then $(i,k) = -1$ and $(j,k) = 1$. In an undirected graph, both entries will be set to 1 instead. For the graph in Figure 8.4, the adjacency and incidence matrices are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad \mathbf{I} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

8.3 Subspaces

A refresher on row and column spaces, linear independence, basis, rank, and orthogonal matrices. To begin:

Definition 8.3.1 — Subspace. A *subspace* $\mathcal{V}$ is a subset of $\mathbb{R}^n$ such that

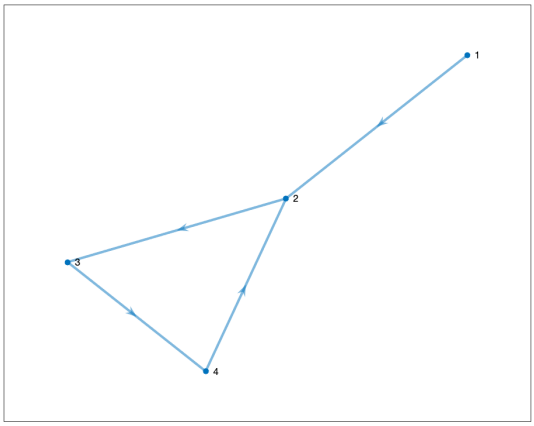


Figure 8.4: A small directed graph.

- $\mathbf{0} \in \mathscr{V}$, and
- If $\mathbf{x}, \mathbf{y} \in \mathscr{V}$ and $\alpha \in \mathbb{R}$, then $\alpha\mathbf{x} + \mathbf{y} \in \mathscr{V}$.

■ **Example 8.2** The set $\mathbb{R}^n$ is a subspace of $\mathbb{R}^n$, as is the set $\{\mathbf{0}\}$ containing only the zero vector. ■

Definition 8.3.2 — Affine Space. An set $\mathscr{W} \in \mathbb{R}^n$ is an *affine space* if there exists a subspace $\mathscr{V}$ and a vector $\mathbf{a} \in \mathbb{R}^n$ such that $\mathscr{W} = \{\mathbf{a} + \mathbf{v} \ : \ \mathbf{v} \in \mathscr{V}\}$. Intuitively, an affine space is a subspace that has been translated, but $\mathscr{W}$ does not contain the zero vector then it is not closed under addition.

■ **Example 8.3** The set of pairs $[x, y]^T \in \mathbb{R}^2$ solving $3x - y = 0$ is a subspace. The set solving $3x - y = 1$ is an affine space, which is equal to the previous space translated by the vector $(0, -1)$. This affine space does not contain $(0,0)$, and it is not closed under addition or scalar multipication since $(0, -1)$ and $(1/3, 0)$ are in the space but $(\frac{1}{3}, -1)$ and $(0, \alpha)$ (for $\alpha \neq -1$) are not. The set of solutions to $3x - y \leq 0$ is neither a subspace nor an affine space. ■

Definition 8.3.3 The *span* of a set of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{R}^n$ is the set of all linear combinations of those vectors:

$$\text{Span}\{\mathbf{a}_1, \ldots, \mathbf{a}_k\} = \{\beta_1\mathbf{a}_1 + \cdots + \beta_k\mathbf{a}_k \ : \ \beta_1, \ldots, \beta_k \in \mathbb{R}\}.$$

■ **Example 8.4** The span of a set of vectors in $\mathbb{R}^n$ is necessarily a subspace of $\mathbb{R}^n$. ■

Given a matrix $\mathbf{A}$, we can define four subspaces of particular interest related to its rows and columns.

Definition 8.3.4 — Four fundamental spaces. The *column space* of a matrix $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$ is

$$\mathscr{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} \ : \ \mathbf{x} \in \mathbb{R}^n\} = \text{Span}\{\mathbf{a}_1, \ldots, \mathbf{a}_n\},$$

i.e., the set of all linear combinations of the columns of $\mathbf{A}$. It is a subspace of $\mathbb{R}^m$.

The *row space* of $\mathbf{A}$ is the set of all linear combinations of the rows of $\mathbf{A}$,

$$\mathscr{R}(\mathbf{A}^T) = \{\mathbf{A}^T\mathbf{y} \ : \ \mathbf{y} \in \mathbb{R}^m\},$$

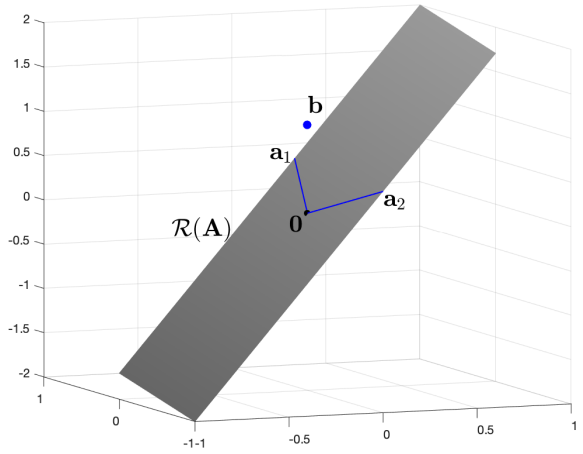


Figure 8.5: The vector $\mathbf{b}$ lies outside the column space of $\mathbf{A}$.

and is a subspace of $\mathbb{R}^n$.

The *null space* of $\mathbf{A}$ is the set of all vectors $\mathbf{z} \in \mathbb{R}^n$ such that $\mathbf{Az} = \mathbf{0}$,

$$\mathcal{N}(\mathbf{A}) = \{\mathbf{z} \,:\, \mathbf{Az} = \mathbf{0}\},$$

and is a subspace of $\mathbb{R}^n$.

The *left null space* of $\mathbf{A}$ is the null space of the transpose, $\mathcal{N}(\mathbf{A}^T)$, and is a subspace of $\mathbb{R}^m$.

These subspaces are relevant when discussing the sets of solutions to linear systems. The next theorem is an immediate consequence of the definition of a column space.

Theorem 8.3.1 For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and vector $\mathbf{b} \in \mathbb{R}^m$, the system $\mathbf{Ax} = \mathbf{b}$ has a solution if and only if $\mathbf{b} \in \mathscr{R}(\mathbf{A})$.

Furthermore, the set of all solutions to a given linear system is an affine space closely related to the null space.

Theorem 8.3.2 If $\mathbf{x}_0$ is a vector satisfying $\mathbf{Ax}_0 = \mathbf{b}$, then the set of all solutions to the system $\mathbf{Ax} = \mathbf{b}$ is the affine space $\mathbf{x}_0 + \mathcal{N}(\mathbf{A})$.

Proof. Using the fact that $\mathbf{Ax}_0 = \mathbf{b}$,

$$\{\mathbf{x} \,:\, \mathbf{Ax} = \mathbf{b}\} = \{\mathbf{x} \,:\, \mathbf{Ax} = \mathbf{Ax}_0\} = \{\mathbf{x} \,:\, \mathbf{A}(\mathbf{x}-\mathbf{x}_0) = \mathbf{0}\} = \{\mathbf{x}_0+\mathbf{z} \,:\, \mathbf{Az} = \mathbf{0}\} = \mathbf{x}_0 + \mathcal{N}(\mathbf{A}).$$

■

■ **Example 8.5** A particle can move in $\mathbb{R}^3$ along the directions $[1,3,1]^T$ or $[0,-2,1]^T$. If it starts at the origin, is the particle capable of reaching the point $[0,0,1]^T$?

One way to interpret this problem is to define

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 3 & -2 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

and ask whether $\mathbf{b} \in \mathscr{R}(\mathbf{A})$. Figure 8.5 shows that it does not. ■

8.3.1 Linear Independence

Definition 8.3.5 — Linear independence. A set of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{R}^n$ is *linearly independent* if the only solution to the system

$$\beta_1 \mathbf{a}_1 + \cdots + \beta_k \mathbf{a}_k = \mathbf{0}$$

is $\beta_1 = \cdots = \beta_k = 0$. The set is *linearly dependent* if it is not linearly independent.

■ **Example 8.6** The vectors

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

are linearly independent. ■

If a set of vectors is linearly dependent, then one of the vectors can be written as a linear combination of the others: pick an index i for which $\beta_i \neq 0$, and get

$$\mathbf{a}_i = (-\beta_1/\beta_i)\mathbf{a}_1 + \cdots + (-\beta_{i-1}/\beta_i)\mathbf{a}_{i-1} + (-\beta_{i+1}/\beta_i)\mathbf{a}_{i+1} + \cdots + (-\beta_k/\beta_i)\mathbf{a}_k.$$

The converse is also true: if one vector is a linear combination of the others, then the set is linearly dependent.

■ **Example 8.7** The vectors

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, \quad \mathbf{a}_3 = \begin{bmatrix} 5 \\ 6 \\ 1 \end{bmatrix}$$

are linearly dependent since $\mathbf{a}_3 = \mathbf{a}_1 + 2\mathbf{a}_2$. ■

■ **Example 8.8** A list containing a single vector is linearly independent if and only if the vector is nonzero. ■

■ **Example 8.9** Any list of vectors containing the zero vector is linearly dependent. ■

■ **Example 8.10** A list of two vectors is linearly dependent if and only if one is a scalar multiple of the other (i.e., the vectors are parallel). ■

■ **Example 8.11** The standard basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are linearly independent. ■

8.3.2 Basis

Recalling Definition 8.3.3, we say that a set of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{R}^n$ *spans* a subspace $\mathscr{V}$ if $\text{Span}\{\mathbf{a}_1, \ldots, \mathbf{a}_k\} = \mathscr{V}$. If the set of vectors is also linearly independent, then we say that it is a *basis* for $\mathscr{V}$.

Definition 8.3.6 — Basis. A *basis* of a subspace $\mathscr{V}$ is a set of linearly independent vectors spanning $\mathscr{V}$.

If $\{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$ is a basis for $\mathscr{V}$, then any vector $\mathbf{x} \in \mathscr{V}$ can be written as a *unique* linear combination of the basis vectors. To see this, suppose that we have two linear combinations

$$\mathbf{x} = \beta_1 \mathbf{a}_1 + \cdots + \beta_k \mathbf{a}_k = \gamma_1 \mathbf{a}_1 + \cdots + \gamma_k \mathbf{a}_k.$$

Subtracting coefficients tells us that $\mathbf{0} = (\beta_1 - \gamma_1)\mathbf{a}_1 + \cdots + (\beta_k - \gamma_k)\mathbf{a}_k$. Since the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are linearly independent it follows that all coefficients are zero, and therefore that $\beta_i = \gamma_i$ for all $1 \leq i \leq k$.

Crucially, any two bases of a vector space or subspace must have the same number of elements.

Theorem 8.3.3 — Dimension Theorem. Given a subspace $\mathscr{V}$ of $\mathbb{R}^n$, any two bases of $\mathscr{V}$ have the same number of elements.

This fact allows us to define the *dimension* of a space.

Definition 8.3.7 — Dimension. The *dimension* of a subspace $\mathscr{V}$, denoted $\dim(\mathscr{V})$, is the number of vectors in any basis of $\mathscr{V}$.

■ **Example 8.12** The standard basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ form a basis for $\mathbb{R}^n$, and so $\mathbb{R}^n$ is n-dimensional. The only n-dimensional subspace of $\mathbb{R}^n$ is $\mathbb{R}^n$ itself, but it has many different possible bases. ■

■ **Example 8.13** The zero space $\{\mathbf{0}\}$ is zero-dimensional. ■

■ **Example 8.14** The space given in Example 8.3 is 1-dimensional, since it is spanned by the single vector $[1,3]^T$. ■

■ **Example 8.15** The formal term "dimension" is meant to align with the common usage of the word. A 1-dimensional space is a line extending infinitely in all directions. A 2-dimensional space is a flat plane extending infinitely in all directions. Figure 8.5 depicts a portion of a 2-dimensional space, and a point not contained in that space. ■

■ **Example 8.16** Model a ball of radius 1 as a set of points in $\mathbb{R}^3$: for a given center $\mathbf{c} \in \mathbb{R}^3$, $S = \{\mathbf{x} : \|\mathbf{x} - \mathbf{c}\|_2 \leq 1$. This ball is a 3-dimensional object, but it is not a subspace because it is not closed under addition or scalar multiplication. ■

8.3.3 Rank

Armed with these definitions, we can return our attention to matrices.

Definition 8.3.8 — Rank. The *rank* of a matrix $\mathbf{A}$ is the dimension of its column space:

$$\operatorname{rank}(\mathbf{A}) = \dim(\mathscr{R}(\mathbf{A})).$$

As it turns out, the rank of a matrix is *also* equal to the dimension of its row space. More generally, we have the following theorem.

Theorem 8.3.4 — Fundamental Theorem of Linear Algebra. Let the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ have rank r. Then

- $\dim(\mathscr{R}(\mathbf{A})) = r$;
- $\dim(\mathscr{R}(\mathbf{A}^T)) = r$;
- $\dim(\mathscr{N}(\mathbf{A})) = n - r$;
- $\dim(\mathscr{N}(\mathbf{A}^T)) = m - r$.

■ **Example 8.17** What is the rank of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 8 & 10 & 12 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}?$$

The rank is equal to the dimension of the row space. Since there are only three rows, the rank must therefore be 0, 1, 2, or 3. The second and third rows are linearly independent since they are not parallel, so the rank must be at least 2. The first and second rows *are* parallel, however, so the rank cannot be 3. Therefore, $\operatorname{rank}(\mathbf{A}) = 2$. ■

Theorem 8.3.5 For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\text{rank}(\mathbf{A}^T\mathbf{A}) = \text{rank}(\mathbf{A})$.

Proof. If $\mathbf{Ax} = \mathbf{0}$, then $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{0} = \mathbf{0}$. Conversely, if $\mathbf{A}^T\mathbf{Ax} = \mathbf{0}$, then

$$0 = \mathbf{x}^T\mathbf{0} = \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} = \|\mathbf{Ax}\|_2^2,$$

which implies that $\mathbf{Ax} = 0$. Thus $\mathscr{N}(\mathbf{A}^T\mathbf{A}) = \mathscr{N}(\mathbf{A})$, and by Theorem 8.3.4 it follows that they have the same rank. ■

Theorem 8.3.6 A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is invertible if and only if $\text{rank}(\mathbf{A}) = n$.

Definition 8.3.9 A matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ has *full column rank* if $\text{rank}(\mathbf{A}) = n$, meaning its columns are linearly independent. It has *full row rank* if $\text{rank}(\mathbf{A}) = m$, meaning its rows are linearly independent. It has *full rank* if $\text{rank}(\mathbf{A}) = \min\{m,n\}$. Since $\text{rank}(\mathbf{A}) \leq \min\{m,n\}$ always, this last definition means that $\text{rank}(\mathbf{A})$ is as large as it can be considering its dimensions.

Fact 8.3.7 If $\mathbf{A} \in \mathbb{R}^{m\times n}$ has full row rank, then the system $\mathbf{Ax} = \mathbf{b}$ has at least one solution because $\mathscr{R}(\mathbf{A}) = \mathbb{R}^m$.

Fact 8.3.8 If $\mathbf{A}$ has full column rank, then the system $\mathbf{Ax} = \mathbf{b}$ has at most one solution because $\mathscr{N}(\mathbf{A}) = \{\mathbf{0}\}$.

Definition 8.3.10 If $\mathbf{A} \in \mathbb{R}^{m\times n}$ where $m > n$, then $\mathbf{A}$ is a *tall-skinny* matrix and the system $\mathbf{Ax} = \mathbf{b}$ is said to be *overdetermined* (because the number of constraints m to the system is greater than the number of free parameters n in $\mathbf{x}$.)

If $m < n$ instead, then $\mathbf{A}$ is a *short-fat* matrix and the system $\mathbf{Ax} = \mathbf{b}$ is said to be *underdetermined*.

8.3.4 Orthogonality

A collection of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k$ is *orthogonal* if $\mathbf{a}_i^T\mathbf{a}_j = 0$ whenever $i \neq j$. If each of these vectors also has 2-norm equal to 1, the collection is *orthonormal* (since a *normalized* vector is one that has been scaled to have norm 1).

■ **Example 8.18** The standard basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ form an orthonormal basis for $\mathbb{R}^n$. ■

Fact 8.3.9 Any collection of orthogonal vectors (all of which are nonzero) is linearly independent.

■ **Example 8.19** The vectors

$$\mathbf{a}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} \frac{3}{5} \\ 0 \\ \frac{4}{5} \end{bmatrix}, \quad \mathbf{a}_3 = \begin{bmatrix} -\frac{4}{5} \\ 0 \\ \frac{3}{5} \end{bmatrix}$$

form an orthonormal basis for $\mathbb{R}^3$. ■

Of great interest are matrices whose columns form an orthonormal set of vectors.

Definition 8.3.11 — Orthogonal Matrix. A matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$ whose columns $\mathbf{q}_1, \ldots, \mathbf{q}_n$ are

orthonormal is called an *orthogonal matrix*. It satisfies the relations

$$\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}_n.$$

In other words, $\mathbf{Q}$ is invertible and $\mathbf{Q}^{-1} = \mathbf{Q}^T$.

Fact 8.3.10 If $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is orthogonal, then so is $\mathbf{Q}^T$. This implies that the rows of $\mathbf{Q}$ are orthonormal as well as the columns.

It is common to use the letters $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{U}$ and $\mathbf{V}$ to denote orthogonal matrices.

R In the event that a tall-skinny matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$ ($m > n$) has orthonormal columns, some books also refer to $\mathbf{Q}$ as an orthogonal matrix. Others consider this terminology confusing, and so we will stick to calling $\mathbf{Q}$ a *matrix with orthonormal columns*.

A tall-skinny matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$ with orthonormal columns will satisfy $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_n$, but on the other hand $\mathbf{Q}\mathbf{Q}^T \neq \mathbf{I}_m$. One explanation for this fact is that $\text{rank}(\mathbf{Q}) = n < m$, so the rank of $\mathbf{Q}\mathbf{Q}^T$ has to be at most n (it is exactly n, in fact).

R If a matrix $\mathbf{A} = \begin{bmatrix}\mathbf{a}_1 & \cdots & \mathbf{a}_n\end{bmatrix} \in \mathbb{R}^{m\times n}$ has *orthogonal* columns, then $\mathbf{A}^T\mathbf{A} = \mathbf{D}$, a diagonal matrix whose diagonal entries are not necessarily equal to 1. If all columns of $\mathbf{A}$ are nonzero, then they can be scaled as $\mathbf{A}\mathbf{D}^{-1/2}$ to have orthonormal columns.

One nice feature about orthonormal bases is that it is relatively simple to write vectors as linear combinations of the basis vectors.

Theorem 8.3.11 Let $\mathbf{Q} = \begin{bmatrix}\mathbf{q}_1 & \cdots & \mathbf{q}_n\end{bmatrix} \in \mathbb{R}^{m\times n}$ have orthonormal columns. For any $\mathbf{x} \in \mathscr{R}(\mathbf{Q})$, the unique vector $\mathbf{y} \in \mathbb{R}^n$ satisfying $\mathbf{x} = \mathbf{Q}\mathbf{y}$ is given by $\mathbf{y} = \mathbf{Q}^T\mathbf{x}$.

Proof. Since $\mathbf{x} \in \mathscr{R}(\mathbf{Q})$, there exists by definition a $\mathbf{y} \in \mathbb{R}^n$ satisfying $\mathbf{x} = \mathbf{Q}\mathbf{y}$. Then multiplying left and right sides by $\mathbf{Q}^T$ gives

$$\mathbf{Q}^T\mathbf{x} = \mathbf{Q}^T\mathbf{Q}\mathbf{y} = \mathbf{I}_n\mathbf{y} = \mathbf{y}.$$

■

Put another way, in terms of the vectors rather than matrices:

Corollary 8.3.12 Let $\mathbf{q}_1, \ldots, \mathbf{q}_n$ be an orthonormal set of vectors in $\mathbb{R}^m$. If $\mathbf{x} \in \text{Span}\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$, then

$$\mathbf{x} = (\mathbf{q}_1^T\mathbf{x})\mathbf{q}_1 + (\mathbf{q}_2^T\mathbf{x})\mathbf{q}_2 + \cdots + (\mathbf{q}_n^T\mathbf{x})\mathbf{q}_n.$$

Reflect and Rotate

So why are matrices with orthonormal columns so important? One key fact about them is that they preserve 2-norms, which means they also preserve distances and angles.

Theorem 8.3.13 Let $\mathbf{Q} \in \mathbb{R}^{m\times n}$ have orthonormal columns. Then the following hold for any $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$:

- $\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$.
- $\|\mathbf{Q}\mathbf{x} - \mathbf{Q}\mathbf{y}\|_2 = \|\mathbf{x} - \mathbf{y}\|_2$.

- $\theta(\mathbf{Qx}, \mathbf{Qy}) = \theta(\mathbf{x}, \mathbf{y})$, where

$$\theta(\mathbf{x}, \mathbf{y}) = \cos^{-1}\left(\frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2}\right)$$

  is the angle between $\mathbf{x}$ and $\mathbf{y}$.

Proof. For the first claim, we use the 2-norm property $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T\mathbf{x}}$ and the fact that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_n$ to get

$$\|\mathbf{Qx}\|_2 = \sqrt{(\mathbf{Qx})^T\mathbf{Qx}} = \sqrt{\mathbf{x}^T\mathbf{Q}^T\mathbf{Qx}} = \sqrt{\mathbf{x}^T\mathbf{I}_n\mathbf{x}} = \sqrt{\mathbf{x}^T\mathbf{x}} = \|\mathbf{x}\|_2.$$

The other two claims can be shown in a similar manner. ■

Interestingly, a converse of this theorem also holds: orthogonal matrices are the only ones that preserve all lengths. The proof is a little more involved, so we omit it here.

Fact 8.3.14 If a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ has the property that $\|\mathbf{Ax}\|_2 = \|\mathbf{x}\|_2$ for all $\mathbf{x} \in \mathbb{R}^n$, then $\mathbf{A}$ is orthogonal.

The norm-preservation property also extends to the matrix 2-norm and Frobenius norm.

Fact 8.3.15 For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and matrices $\mathbf{Q} \in \mathbb{R}^{\ell\times m}$, $\mathbf{P} \in \mathbb{R}^{n\times p}$ with orthonormal columns, $\|\mathbf{QAP}^T\|_2 = \|\mathbf{A}\|_2$ and $\|\mathbf{QAP}^T\|_F = \|\mathbf{A}\|_F$.

It's also worth noting the norms of these matrices themselves.

Fact 8.3.16 For a matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$ with orthonormal columns, $\|\mathbf{Q}\|_2 = 1$ and $\|\mathbf{Q}\|_F = \sqrt{n}$.

Proof. The 2-norm follows from the fact that $\|\mathbf{Qx}\|_2 = \|\mathbf{x}\|_2$ always. The F-norm follows from the fact that each of the n columns of $\mathbf{Q}$ have 2-norm equal to 1. ■

Here are a few particularly valuable types of orthogonal matrices:

Definition 8.3.12 — Rotation matrix. A *rotation matrix* for $\mathbb{R}^2$ has the form

$$\mathbf{G}_\theta = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix},$$

and has the effect of rotating a point $[x, y]^T \in \mathbb{R}^2$ counterclockwise by an angle of θ. More generally, a *Givens rotation* for $\mathbb{R}^n$ is a rotation of two of the coordinate axes, such as

$$\mathbf{G} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\theta) & -\sin(\theta)\\ 0 & \sin(\theta) & \cos(\theta)\end{bmatrix},$$

which leaves the x-axis fixed while rotating the y- and z-axes.

Definition 8.3.13 — Householder reflections. Given a unit vector $\mathbf{u} \in \mathbb{R}^n$ ($\|\mathbf{u}\|_2 = 1$, see Definition 3.4.3), a *Householder reflection* has the form

$$\mathbf{H}_\mathbf{u} = \mathbf{I}_n - 2\mathbf{uu}^T,$$

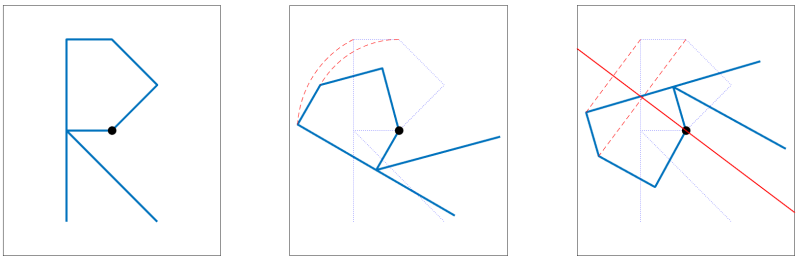


Figure 8.6: Center: rotation clockwise by 60 degrees. Right: reflection across the plane normal to the vector $\mathbf{u} = [0.6, 0.8]^T$ (red dashed lines are parallel to $\mathbf{u}$). The black circle represents the origin in all three images.

and has the effect of reflecting a vector across the hyperplane normal to $\mathbf{u}$.

This means that in 2 dimensions, $\mathbf{H_u}$ reflects across the line perpendicular to $\mathbf{u}$. In 3 dimensions, it reflects across the 2D plane normal to $\mathbf{u}$.

■ **Example 8.20** Figure 8.6 demonstrates the effects of rotation by $\frac{\pi}{3}$ radians (60 degrees) and reflection across the plane normal to the vector $\mathbf{u} = [0.6, 0.8]^T$. These orthogonal transformations are represented by the matrices

$$\mathbf{G}_{\pi/3} = \frac{1}{2}\begin{bmatrix} 1 & -\sqrt{3} \\ \sqrt{3} & 1 \end{bmatrix}, \qquad \mathbf{H_u} = \frac{1}{25}\begin{bmatrix} 7 & -24 \\ -24 & -7 \end{bmatrix}$$

■

R Givens rotations will always have a determinant of 1, and Householder reflections will always have a determinant of -1. This is related to the fact that Givens rotations preserve *orientation* while the Householder reflections do not.

Definition 8.3.14 — Permutation matrices. A *permutation matrix* is a square matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$ with a single 1 in each row and column and zeros elsewhere. It has the effect of permuting the coordinate indices of a vector:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_5 \\ x_3 \\ x_4 \\ x_2 \end{bmatrix}.$$

A permutation matrix can be described by a single vector $\mathbf{p} \in \mathbb{R}^n$: in the above case, $\mathbf{p} = [1, 5, 3, 4, 2]^T$. This indicates that $\mathbf{P}_{i,\mathbf{p}(i)} = 1$ for each $1 \leq i \leq n$, in which case it will hold in Matlab that `Px = x(p)`. Thus permutation matrices are significantly cheaper to store and multiply with than other matrices of the same size.

8.4 Matlab Commands

Many useful commands related to forming and manipulating vectors and matrices are presented in Chapter 3.

Rank

- `rank(A)`: Estimates the number of linearly independent rows/columns of `A`. Singular values of `A` below a certain threshold will not be counted. `rank(A,TOL)` allows the user to set the threshold.
- `det(A)`: The determinant of `A`.
- `orth(A)`: An orthonormal basis for the range of `A`.
- `null(A)`: An orthonormal basis for the null space of `A`. If `A` is a small matrix with integer entries, `null(A,'r')` returns a basis whose elements are ratios of small integers.
- `rref(A)`: The reduced row echelon form of `A`. If `A` is invertible, then the final column of `rref([A,b])` is equal to $A^{-1}b$. However, Matlab has more efficient ways to solve this problem.

Solving Linear Systems

- `A\B`: If `A` is invertible, returns $A^{-1}b$ (but computed in a different way). If `A` is not invertible, returns a solution to the least-squares problem $\min_X \|AX - B\|_2$.
- `B/A`: If `A` is invertible, returns BA^{-1}. More generally, `A/B = (B'\A')'`.
- `inv(A)`: The inverse of `A`. *Note: if all you want to do is solve a system of linear equations, it is inefficient to compute the inverse explicitly! Use the backslash operation instead.*

Factorizations

- `[L,U] = lu(A)`: Returns the LU factorization of `A` (i.e., performs Gaussian elimination). `L` and `U` are respectively lower and upper triangular matrices satisfying `A = LU`.
- `R = chol(A)`: Returns the Cholesky factorization of `A`. `R` is an upper triangular matrix such that `A = R'*R`. Use instead of `lu` when `A` is symmetric with all eigenvalues strictly positive.
- `[L,D] = ldl(A)`: Returns the block LDL^T factorization of `A`. `D` is block diagonal and `L` is a permuted lower triangular matrix such that `A = LDL'`. Use instead of `lu` when `A` is symmetric but does not have all positive eigenvalues.
- `[Q,R]` = qr(A): Returns a square orthogonal matrix `Q` and upper triangular matrix `R` satisfying `A = QR`. For the "economy-size" decomposition, use `qr(A,0)`.

Timing

- tic: Start a stopwatch timer. `tstart = tic` saves a time to variable `tstart`.
- toc: Read the timer started by `tic`. `telapsed = toc(tstart)` will save the elapsed time.

9. Rank and Conditioning

Here we explore the concept of rank further. In particular, we address the question of how all of the material from the previous section behave in the presence of error. What does it mean for a matrix to be *almost* singular? How do we measure the rank of a matrix if the entries themselves are subject to some uncertainty?

9.1 Matrix Factorizations

In general, a *matrix decomposition* or *factorization* is a factorization of a matrix into a product of matrices $\mathbf{A} = \mathbf{F}_1 \cdots \mathbf{F}_k$. There are many different popular factorizations:

- For a square matrix $\mathbf{A}$, the *PLU* factorization finds a factorization $\mathbf{A} = \mathbf{PLU}$ where $\mathbf{P}$ is a permutation matrix and $\mathbf{L}$ and $\mathbf{U}$ are respectively lower and upper triangular. This factorization is closely related to Gaussian elimination and useful when solving linear systems.
- If $\mathbf{A}$ is diagonalizable, the eigenvalue decomposition finds an invertible matrix $\mathbf{V}$ and diagonal matrix $\mathbf{D}$ so that $\mathbf{A} = \mathbf{VDV}^{-1}$. The columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{A}$ and the diagonal elements of $\mathbf{D}$ are the eigenvalues.
- If $\mathbf{A}$ is a real symmetric matrix with all positive eigenvalues, the *Cholesky factorization* solves $\mathbf{A} = \mathbf{LL}^T$, where $\mathbf{L}$ is lower triangular (as mentioned in Definition 8.1.3).
- The *QR* factorization finds $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ has orthonormal columns and $\mathbf{R}$ is upper triangular.
- The *singular value decomposition* of $\mathbf{A} \in \mathbb{R}^{m \times n}$ finds $\mathbf{A} = \mathbf{U\Sigma V}^T$, where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m \times n}$ has decreasing nonnegative real numbers $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ on its main diagonal and zeros elsewhere. We'll cover this one in much more detail later.

We focus for now on the following general type of factorization.

Definition 9.1.1 A *rank factorization* of a rank-r matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a factorization $\mathbf{A} = \mathbf{LR}$, where $\mathbf{L} \in \mathbb{R}^{m \times r}$ and $\mathbf{R} \in \mathbb{R}^{r \times n}$.

Fact 9.1.1 Every matrix has a rank factorization.

The rank of a matrix is the *smallest* number r such that a factorization $\mathbf{LR}$ of inner dimension r exists.

Theorem 9.1.2 If $\mathbf{A} \in \mathbf{R}^{m\times n}$ has a factorization $\mathbf{LR}$ with $\mathbf{L} \in \mathbb{R}^{m\times r}$ and $\mathbf{R} \in \mathbb{R}^{r\times n}$, then $\text{rank}(\mathbf{A}) \leq r$. If both $\mathbf{L}$ and $\mathbf{R}$ have rank r, then so does $\mathbf{A}$.

Proof. It can be checked that $\mathscr{R}(\mathbf{A}) \subseteq \mathscr{R}(\mathbf{L})$, so $\text{rank}(\mathbf{A}) \leq \text{rank}(\mathbf{L}) \leq r$. For the second claim, if $\mathbf{R}$ has rank r then $\mathscr{R}(\mathbf{R}) = \mathbb{R}^r$, so

$$\mathscr{R}(\mathbf{A}) = \mathscr{R}(\mathbf{LR}) = \mathscr{R}(\mathbf{L})$$

and therefore $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{L}) = r$. ■

As intuition for this theorem, think of $\mathbf{A}$ as a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$. The factorization shows that $\mathbf{A}$ can be expressed as a *composition* of two linear transformations, one from $\mathbb{R}^n$ to $\mathbb{R}^r$ and the second from $\mathbb{R}^r$ to $\mathbb{R}^m$. Assuming $r \leq \min\{m,n\}$, this means that an n-dimensional space is *compressed* down to an r-dimensional space, which is then *embedded* in an m-dimensional space. Crucially, the second transformation $\mathbf{L}$ cannot increase the dimension of the image: linear transformations can shrink, but never grow, the dimensions of subspaces, and once the "information" is lost it cannot be recovered again.

Fact 9.1.3 For any subspace $\mathscr{V}$ of $\mathbb{R}^n$ and linear transformation $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}^m$, $\dim(\mathbf{A}\mathscr{V}) \leq \dim(\mathscr{V})$.

A related implication is that the rank of a product of matrices cannot be greater than the smallest rank of a matrix in the product.

Theorem 9.1.4 $\text{rank}(\mathbf{AB}) \leq \min\{\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B})\}$

Proof. Take a rank factorization $\mathbf{A} = \mathbf{LR}$, and get $\mathbf{AB} = \mathbf{L}(\mathbf{RB})$. By Theorem 9.1.2, $\text{rank}(\mathbf{AB}) \leq \text{rank}(\mathbf{L}) = \text{rank}(\mathbf{A})$. A similar strategy shows that $\text{rank}(\mathbf{AB}) \leq \text{rank}(\mathbf{B})$. ■

The rank factorization can similarly be used to bound the rank of sums and concatenations of matrices.

Fact 9.1.5 For any $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$,

$$\text{rank}(\mathbf{A}+\mathbf{B}) \leq \text{rank}(\begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}) \leq \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B}).$$

9.1.1 CX Decomposition

One specific type of rank factorization is the CX decomposition.

Definition 9.1.2 The *CX decomposition* of a rank-r matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a factorization $\mathbf{A} = \mathbf{CX}$, where $\mathbf{C} = \begin{bmatrix}\mathbf{a}_{k_1} & \cdots & \mathbf{a}_{k_r}\end{bmatrix}$ is an $m \times r$ matrix whose columns are a subset of the columns of $\mathbf{A}$.

The idea is that if $\text{rank}(\mathbf{A}) = r$, then there is a collection of r linearly independent columns of $\mathbf{A}$ (not necessarily unique) that spans $\mathscr{R}(\mathbf{A})$. This means that all of the columns of $\mathbf{A}$ can be written as (unique) linear combinations of these r columns, and the matrix $\mathbf{X}$ describes these linear combinations: for each $1 \leq j \leq n$,

$$\mathbf{a}_j = \mathbf{C}\mathbf{x}_j = x_{1j}\mathbf{c}_1 + \cdots + x_{rj}\mathbf{c}_r = x_{1j}\mathbf{a}_{k_1} + \cdots x_{rj}\mathbf{a}_{j_r}.$$

A similar factorization in terms of the rows of $\mathbf{A}$ can be obtained by e.g., taking the CX decomposition of $\mathbf{A}^T$.

■ **Example 9.1 — Relation to Reduced Echelon Form.** It is possible to find a CX factorization by computing the reduced echelon form (Matlab: `rref`) of a matrix. For example, the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 1 & 4 \\ 2 & 7 & 3 & 9 \\ 1 & 5 & 3 & 1 \\ 1 & 2 & 0 & 8 \end{bmatrix} \sim \begin{bmatrix} \underline{1} & 0 & -2 & 0 \\ 0 & \underline{1} & 1 & 0 \\ 0 & 0 & 0 & \underline{1} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

has rank 3, and its reduced echelon form has pivots (underlined) in columns 1, 2, and 4. We can then set $\mathbf{C}$ as the corresponding columns in $\mathbf{A}$ and $\mathbf{X}$ as the first three rows of the reduced echelon form:

$$\mathbf{C} = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 7 & 9 \\ 1 & 5 & 1 \\ 1 & 2 & 8 \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} 1 & 0 & -2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

■

The CX decomposition has a useful interpretation in data science applications: if all columns of $\mathbf{A}$ are linear combinations of the subset in $\mathbf{C}$ and the columns represent attributes and the rows represent objects, then all of the attributes in $\mathbf{A}$ can be "explained" by the ones in $\mathbf{C}$. In example 9.1, suppose that the rows represent people and the columns represent the ratings they gave to four movies. The CX decomposition implies that a person's rating for movie 3 can be predicted perfectly from their ratings for movies 1 and 2. Specifically, $\mathbf{a}_3 = -2\mathbf{a}_1 + \mathbf{a}_2$.

It can also be useful to think of rank factorizations in terms of *outer products*.

Definition 9.1.3 Given vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^m$, the *outer product* of $\mathbf{u}$ and $\mathbf{v}$ is the $m \times n$ matrix

$$\mathbf{u}\mathbf{v}^T = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_nv_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{bmatrix} \in \mathbb{R}^{m \times n},$$

where the notation $\mathbf{u}\mathbf{v}^T$ interprets $\mathbf{u}$ as a $m \times 1$ matrix multiplied by the $1 \times n$ matrix $\mathbf{v}^T$.

If a matrix has the factorization $\mathbf{A} = \mathbf{U}\mathbf{V}^T$ where $U \in \mathbb{R}^{m \times r}$ and $\mathbf{V} \in \mathbf{n} \times \mathbf{r}$, then it can also be written as the sum of r outer products

$$\mathbf{A} = \mathbf{U}\mathbf{V}^T = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_r \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_r \end{bmatrix}^T = \sum_{i=1}^{r} \mathbf{u}_i \mathbf{v}_i^T.$$

Fact 9.1.6 The rank of a matrix is the smallest number r such that $\mathbf{A}$ can be written as the sum of r outer products.

■ **Example 9.2** The matrix $\mathbf{A}$ from Example 9.1 can be written as the rank-3 sum

$$\mathbf{A} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 & 0 \end{bmatrix} + \begin{bmatrix} 3 \\ 7 \\ 5 \\ 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 4 \\ 9 \\ 1 \\ 8 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}.$$

■

9.1.2 Compression

So what's the significance of a matrix having a low-rank (i.e., $r \ll \min\{m,n\}$) factorization? In conceptual terms, it might mean that your high-dimensional data set can be described accurately by a much smaller one. Your database may have ratings from millions of users for thousands of movies, but if the data matrix is low-rank then there might be a small number of *types* of movies. Once a user's preference for these underlying types is established, their preference for any individual movie could be determined accurately. In this manner, low-rank factorizations can help describe a model with many parameters in terms of a simpler one.

In computational terms, low-rank matrices are cheap to store and manipulate. To store each entry of an $m \times n$ matrix requires mn data, but if the matrix has a rank-r factorization then only $(m+n)r$ data is required. This doesn't help if r is very close to m or n—in Example 9.1, it is more costly to store $\mathbf{C}$ and $\mathbf{X}$ than to just store $\mathbf{A}$ itself. But if $r \ll \min\{m,n\}$, then the savings can be substantial.

Similarly, computing the matrix-vector product $\mathbf{Ax}$ for $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{x} \in \mathbb{R}^n$ requires about $2mn$ flops (mn multiplies and $m(n-1)$ adds). If we have access to $\mathbf{A} = \mathbf{LR}$ in factored form, then computing $\mathbf{L}(\mathbf{Rx})$ will only require about $2(m+n)r$ flops instead. Take care here, as the order of operations matters! The product $(\mathbf{LR})\mathbf{x}$ would require about $2mnr + 2mn$ flops instead, since this ordering first forms the product $\mathbf{LR}$ explicitly, then multiplies it by $\mathbf{x}$.

Note that there are many types of matrices that have a simple structure without necessarily having a small rank. The identity matrix (and more generally, any diagonal matrix with nonzeros on the diagonal) is very simple to describe and do computations with, but has full rank. The same goes for Householder reflections, which have the form $\mathbf{H_u} = \mathbf{I} - 2\mathbf{uu}^T$. They are full rank, but can be described entirely in terms of the vector $\mathbf{u}$.

■ **Example 9.3** Figure 9.1 presents two images of the MIT logo, one of which has been rotated 45 degrees. The second does not look any more "complicated" than the first, but the first can be approximated by a low-rank matrix far more easily than the second. This is because the visible structure in the image aligns in a natural way with the rows and columns of the matrix. Having a simple structure does not necessarily mean having low rank! ■

9.2 Low-Rank Approximation

In practice, we will rarely be so lucky as to encounter a matrix that has a low rank *exactly*. In Figure 9.1, for example, the image of the MIT logo on the left looks like it might have rank 6 or so, since there are only six different types of columns in the matrix: one for the white background, two for the M, one for the I, and two for the T. However, the image contains small variations not visible in this printing–not all of the white pixels are an exact 255 on the [0,255] scale, for example, so columns that appear identical may not actually be so. As a result, Matlab reports that the image matrix has rank 125. On the other hand, consider the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 10^{-20} \end{bmatrix}. \tag{9.1}$$

Matlab will report that the rank of $\mathbf{A}$ is 1, but it clearly has rank 2! So exactly how much can we trust Matlab's computed ranks for the MIT logos?

Informally, the explanation for this is that if a matrix is "very close" one with a smaller rank, then Matlab counts only up to the smaller rank. Formally, it gives this explanation in the documentation for `rank`:

> `rank(A,TOL)` is the number of singular values of `A` that are larger than `TOL`. By default, `TOL = max(size(A)) * eps(norm(A))`.

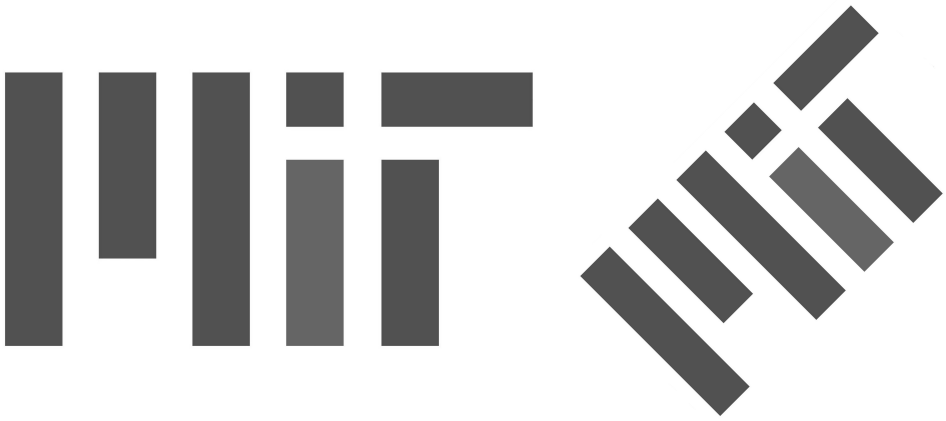


Figure 9.1: Left: MIT logo, size 1574×2800. Right: logo rotated 45 degrees, size 3092×3092.

This might behavior might seem a little strange in the case of 9.1, since the matrix is diagonal and the columns are clearly linearly independent. It makes more sense when no particular structure about the matrix is assumed.

■ **Example 9.4** Consider the matrices

$$\mathbf{A}_1 = \begin{bmatrix} 1+10^{-10} & 1 \\ 1 & 1-10^{-10} \end{bmatrix}, \qquad \mathbf{A}_2 = \begin{bmatrix} 1+10^{-10} & 1-10^{-20} \\ 1 & 1-10^{-10} \end{bmatrix}.$$

The first matrix has rank 2 while the second has rank 1, but in finite precision $\text{fl}(\mathbf{A}_1) = \text{fl}(\mathbf{A}_2)$. ■

The underlying problem is that $\text{rank}(\cdot)$ is a discontinuous function, and small uncertainty in the elements of a matrix or small rounding errors made by the computer can potentially have a large effect on the computed rank. In practice, we use the *numeric rank* of a matrix to refer to the approximate rank of the matrix, using a threshold to decide what it means to be "sufficiently close" to a matrix of smaller rank.

What's a good way to measure how close two matrices are to one another? Norms, of course! As mentioned in Section 3.5 two of our favorite norms are the 2-norm and Frobenius norm

$$\|\mathbf{A}\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\lambda_{\max}(\mathbf{A}^T\mathbf{A})}, \qquad \|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2},$$

and for a given norm we can use $\|\mathbf{A}-\mathbf{B}\|$ to measure the distance between same-sized matrices $\mathbf{A}$ and $\mathbf{B}$. This brings us to the low-rank approximation problem, which commonly comes in two flavors:

- The *fixed-rank* problem: given a target rank r, find a matrix $\widetilde{\mathbf{A}}_r$ minimizing $\|\mathbf{A}-\widetilde{\mathbf{A}}_r\|$.
- The *fixed-accuracy* problem: given a threshold $0 < \tau < 1$, find a matrix $\widetilde{\mathbf{A}}_r$, with rank r as small as possible, satisfying $\|\mathbf{A}-\widetilde{\mathbf{A}}_r\| \leq \tau\|\mathbf{A}\|$.

■ **Example 9.5** Figure 9.2 shows the optimal rank-r truncations of Matlab's clown image. The storage required for a rank-r factorization of an $m \times n$ matrix is $(m+n)r$, and here the rank-20 factorization requries only about 1/6 the storage of the original image. The relative error is measured in terms of the Frobenius norm,

$$\text{relative error} = \frac{\|\mathbf{A}-\widetilde{\mathbf{A}}_r\|_F}{\|\mathbf{A}\|_F}.$$

■

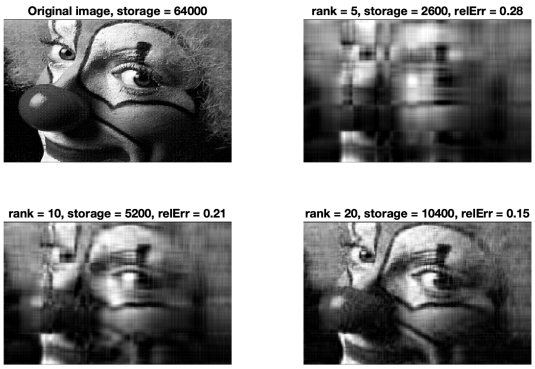


Figure 9.2: Optimal low-rank truncations of grayscale image for $r = 5, 10, 20$. Original image is 200×320.

One very unofficial measure of error is known as the *eyeball norm*, which is said to be small if two images "look like one another". This measure can be rather different from the 2-norm or Frobenius norm error depending on the scenario, so although the latter two are simpler to measure and to minimize, they might not deliver the most satisfactory results.

9.2.1 Geometric Interpretation

What does it mean for a matrix to *almost* have a low-rank factorization? The following equivalent definition of the rank of a matrix might help.

Fact 9.2.1 The rank of a linear transformation $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}^n$ is the dimension of the image of $\mathbf{A}$.

This is almost completely equivalent to saying that the rank of $\mathbf{A}$ is the dimension of $\mathscr{R}(\mathbf{A})$, but the intended mindset is slightly different. The bottom images in Figure 9.3 show the effect of a rank-2 linear transformation on the vertex set of the Utah teapot: all points are sent to a particular 2-dimensional space, so the teapot has been compressed. The fact that this rank-2 matrix in $\mathbb{R}^{3\times 3}$ is not invertible means that this compression cannot be reversed through subsequent linear transformations: some information about the 3-dimensional object has been lost for good. For the record, the two matrices used are

$$\mathbf{A} = \frac{1}{7}\begin{bmatrix} 1 & 2 & 3 \\ 3 & -1 & 1 \\ 2 & 2 & 2 \end{bmatrix}, \qquad \mathbf{B} \approx \frac{1}{7}\begin{bmatrix} 1.19 & 2.30 & 2.66 \\ 3.06 & -0.91 & 0.89 \\ 1.77 & 1.64 & 2.41 \end{bmatrix},$$

the first having full rank and the second having rank 2.

The top two images show the effect of a linear transformation that is *close* to having rank 2, but still has full rank. The teapot has been squashed, but still remains a 3-dimensional object. With that observation in mind, we might try something like the following interpretation:

Definition 9.2.1 — Informal interpretation. Recall from Definition 3.4.3 that the *unit sphere* (for the 2-norm) is the set of all points $\mathbf{x} \in \mathbb{R}^n$ satisfying $\|\mathbf{x}\|_2 = 1$. Denote this sphere by $\mathscr{S}$. Then a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is *close* to having rank r if the set $\mathbf{A}\mathscr{S} = \{\mathbf{Ax} \ : \ \mathbf{x} \in \mathscr{S}\}$ *nearly* lies within an r-dimensional subspace.

What does it mean for this set of points to *nearly* lie within a low-dimensional space? We don't have a formal definition for this yet, but will acquire one soon.

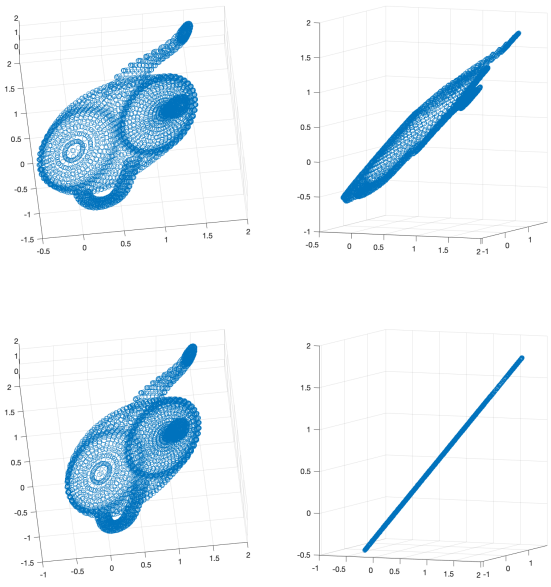


Figure 9.3: Two linear transformations of the Utah teapot vertices, viewed from two different angles. Top: a rank-3 transformation that is "close" to rank 2. Bottom: a rank-2 matrix.

9.2.2 Approximation with Rank Factorization

One potential strategy for solving the low-rank approximation problem is to find and use a rank factorization of the matrix in question. If we have access to the rank factorization of a rank-s matrix

$$\mathbf{A} = \mathbf{U}\mathbf{V}^T = \begin{bmatrix}\mathbf{u}_1 & \cdots & \mathbf{u}_s\end{bmatrix}\begin{bmatrix}\mathbf{v}_1 & \cdots & \mathbf{v}_s\end{bmatrix}^T = \sum_{i=1}^{s}\mathbf{u}_i\mathbf{v}_i^T,$$

then for $r \leq s$ we can obtain a rank-r approximation simply by taking the first r terms $\sum_{i=1}^{r}\mathbf{u}_i\mathbf{v}_i^T$, where the terms $\mathbf{u}_i\mathbf{v}_i^T$ are assumed to be sorted from greatest norm to least.

Some care must be taken here, since not all rank factorizations will work equally well. The following example shows that using the **CX** decomposition naively may yield a poor approximation.

■ **Example 9.6** Consider the matrix

$$\mathbf{A} = \begin{bmatrix}1 & 1.1 & 0\\ 1.1 & 1 & 1\end{bmatrix}, \qquad \|\mathbf{A}\|_2 \approx 2.23$$

Using its reduced row echelon form to compute a CX decomposition gives

$$\mathbf{C} = \begin{bmatrix}1 & 1.1\\ 1.1 & 1\end{bmatrix}, \qquad \mathbf{X} \approx \begin{bmatrix}1 & 0 & 5.24\\ 0 & 1 & -4.76\end{bmatrix}.$$

Truncating this decomposition to get a rank-1 approximation gives

$$\widetilde{\mathbf{A}}_1 \approx \begin{bmatrix}1\\ 1.1\end{bmatrix}\begin{bmatrix}1 & 0 & 5.24\end{bmatrix} \approx \begin{bmatrix}1 & 0 & 5.24\\ 1.1 & 0 & 5.76\end{bmatrix},$$

which makes for a terrible approximation since $\|\mathbf{A} - \widetilde{\mathbf{A}}_1\|_2 \approx 7.23$, which is larger than $\|\mathbf{A}\|_2$ in the first place. The approximation is poor largely because the first two columns—the ones chosen for **C**—are *close* to linearly dependent. ■

One simple (and generally non-optimal) strategy for finding a rank-1 approximation is just to set each entry of $\mathbf{A}$ to the average value of the corresponding column or row.

■ **Example 9.7** Taking the average of each columns of the matrix $\mathbf{A}$ from Example 9.6 gives the rank-1 approximation

$$\widetilde{\mathbf{A}}_1'' = \begin{bmatrix} 1.05 & 1.05 & 0.5 \\ 1.05 & 1.05 & 0.5 \end{bmatrix}.$$

The 2-norm approximation error is $\|\mathbf{A} - \widetilde{\mathbf{A}}_1'\|_2 \approx 0.71$, which is fairly close to optimal. ■

9.3 Sensitivity to Perturbations

Suppose the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is invertible, and that we wish to solve the linear system $\mathbf{Ax} = \mathbf{b}$. Considering the solution $\mathbf{x}$ as a function of the right-hand side $\mathbf{b}$, we might ask how sensitive the solution is to perturbations in the input (or perturbations in $\mathbf{A}$, for that matter).

■ **Example 9.8** Consider the system $\mathbf{Ax} = \mathbf{b}$ with

$$\mathbf{A} = \begin{bmatrix} 1+10^{-4} & 1 \\ 1 & 1+10^{-4} \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1+10^{-4} \\ 1 \end{bmatrix},$$

which has the solution $\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. If we perturb $\mathbf{b}$ slightly to get

$$\tilde{\mathbf{b}} = \begin{bmatrix} 1 \\ 1+10^{-4} \end{bmatrix},$$

then the solution will be perturbed as

$$\tilde{\mathbf{x}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Minor relative perturbations to the right hand side can lead to major perturbations to the solution! ■

Thanks to the fact that $\mathbf{A}$ represents a linear transformation, it is fairly straightforward to bound the sensitivity of the solution to perturbations to $\mathbf{b}$. Writing the perturbed input as $\tilde{\mathbf{b}} = \mathbf{b} + \Delta\mathbf{b}$, we can rewrite the 2-norm relative error in the output as

$$\begin{aligned} \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} &= \frac{\|\mathbf{A}^{-1}\mathbf{b} - \mathbf{A}^{-1}\tilde{\mathbf{b}}\|_2}{\|\mathbf{x}\|_2} \\ &= \frac{\|\mathbf{A}^{-1}\mathbf{b} - \mathbf{A}^{-1}(\mathbf{b} - \Delta\mathbf{b})\|_2}{\|\mathbf{x}\|_2} \\ &= \frac{\|\mathbf{A}^{-1}\Delta\mathbf{b}\|_2}{\|\mathbf{x}\|_2} \\ &= \frac{\|\mathbf{A}^{-1}\Delta\mathbf{b}\|_2}{\|\Delta\mathbf{b}\|_2} \cdot \frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2} \cdot \frac{\|\Delta\mathbf{b}\|_2}{\|\mathbf{b}\|_2} \\ &\leq \|\mathbf{A}^{-1}\|_2 \|\mathbf{A}\|_2 \cdot \frac{\|\Delta\mathbf{b}\|_2}{\|\mathbf{b}\|_2}, \end{aligned}$$

where the fourth equality uses the fact that $\mathbf{Ax} = \mathbf{b}$ and the operator norm $\|\mathbf{A}\|_2$ is as defined in Definition 3.5.3. As a result, we get the following definition of the condition number of a matrix.

Definition 9.3.1 The *condition number* of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ (or, the 2-norm condition number with respect to matrix inversion) is given by

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \|\mathbf{A}^{-1}\|_2.$$

If $\mathbf{A}$ is not invertible then the condition number is defined to be infinite.

Under this definition, condition numbers must be greater than or equal to 1. Furthermore, orthogonal matrices (and no others, up to scaling) have a condition number equal to 1.

Theorem 9.3.1 For any matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\kappa(\mathbf{A}) \geq 1$.

Proof. Since the 2-norm is submultiplicative (Definition 3.5.2), $\|\mathbf{A}\|_2\|\mathbf{A}^{-1}\|_2 \geq \|\mathbf{A}\mathbf{A}^{-1}\|_2 \geq \|\mathbf{I}_n\|_2 = 1$. ■

Before proving the second claim, we offer a useful lemma.

Lemma 9.3.2 For a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\|\mathbf{A}^{-1}\|_2 = \frac{1}{\min_{\mathbf{x}\neq\mathbf{0}} \|\mathbf{A}\mathbf{x}\|_2/\|\mathbf{x}\|_2}$.

Proof. Starting with the definition of the 2-norm and making a substitution $\mathbf{x} = \mathbf{A}\mathbf{y}$ gives

$$\|\mathbf{A}^{-1}\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}^{-1}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{\mathbf{y}\neq\mathbf{0}} \frac{\|\mathbf{y}\|_2}{\|\mathbf{A}\mathbf{y}\|_2} = \left(\min_{\mathbf{y}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{y}\|_2}{\|\mathbf{y}\|_2}\right)^{-1}.$$

■

Instead of considering all $\mathbf{x} \neq \mathbf{0}$ for the definition of the 2-norm, it can be helpful to consider only the set satisfying $\|\mathbf{x}\|_2 = 1$, so that the expression can be simplified somewhat. In any case, with this lemma in hand, we get an interesting interpretation of the condition number:

Fact 9.3.3 For a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$,

$$\kappa(\mathbf{A}) = \frac{\max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2}{\min_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2}.$$

The condition number therefore represents something akin to the *aspect ratio* for a matrix: it is the maximum ratio by which a linear transformation is capable of distorting an image. The squashed teapots in Figure 9.3 give a visual example of this effect: the first matrix has a condition number of about 7. The condition number of the second matrix, which has numerical rank 2, is reported by Matlab as about $6 \cdot 10^{16}$ (note that this is larger than `1/eps`).

Theorem 9.3.4 A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ has condition number 1 if and only if $\mathbf{A} = \alpha\mathbf{Q}$ for some orthogonal matrix $\mathbf{Q}$ and scalar $\alpha \neq 0$.

Proof. First, suppose that $\mathbf{A} = \alpha\mathbf{Q}$ for an orthogonal matrix $\mathbf{Q}$. Then $\|\mathbf{A}\mathbf{x}\|_2 = \|\alpha\mathbf{Q}\mathbf{x}\|_2 = |\alpha|\|\mathbf{x}\|_2$ for every $\mathbf{x} \in \mathbb{R}^n$, so $\|\mathbf{A}\|_2 = |\alpha|$. Similarly, $\|\mathbf{A}^{-1}\mathbf{x}\|_2 = \|\alpha^{-1}\mathbf{Q}^T\mathbf{x}\|_2 = |\alpha^{-1}|$, so $\kappa(\mathbf{A}) = |\alpha\alpha^{-1}| = 1$.

Conversely, suppose that $\kappa(\mathbf{A}) = 1$. Without loss of generality we can scale the matrix as $\mathbf{A} \mapsto \mathbf{A}/\|\mathbf{A}\|_2$ so that $\|\mathbf{A}\|_2 = 1$ since scaling a matrix does not change its condition number (verify this!). It therefore follows from Fact 9.3.3 that $\|\mathbf{A}\mathbf{x}\|_2 = 1$ for every $\|\mathbf{x}\|_2 = 1$, and thus that $\|\mathbf{A}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ for all $\mathbf{x} \in \mathbb{R}^n$. By Fact 8.3.14, it follows that $\mathbf{A}$ is orthogonal. ■

The implication of all this is that orthogonal matrices are the nicest possible matrices for solving systems of linear equations under uncertainty in the data, because they do not amplify the error. Also convenient is the fact that multiplying a matrix by an orthogonal matrix does not change its condition number.

Program 9.1: The Cauchy matrix is poorly conditioned

```
z = 1:8;
C = 1./(z + z'); %cond(A) is about 5.6e10
b = C(:,1) % exact solution is [1,0,...,0]
x = C\b;   % relative error of about 1e-07
```

Fact 9.3.5 For any matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$, $\kappa(\mathbf{QA}) = \kappa(\mathbf{AQ}) = \kappa(\mathbf{A})$.

If a matrix has a large condition number, then rounding errors can have a large effect on the answer even if there is no uncertainty in $\mathbf{A}$ or $\mathbf{b}$.

■ **Example 9.9** The *Cauchy matrix* $\mathbf{C} \in \mathbb{R}^{n\times n}$ has entries

$$c_{ij} = \frac{1}{i+j}, \qquad 1 \le i \le n, 1 \le j \le n.$$

It is invertible for all n, but very badly conditioned. For $n = 8$, we have $\kappa(\mathbf{C}) \approx 5.6 \cdot 10^{10}$. Given the system $\mathbf{Cx} = \mathbf{b}$ where $\mathbf{b} = \mathbf{c}_1$ is set as the first column of $\mathbf{C}$. Program 9.1 does exactly this, and where the answer in exact arithmetic (even accounting for the fact that the entries of $\mathbf{C}$ are rounded) is exactly $\mathbf{e}_1$. The computed answer, however, is

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1.000000000023638 \\ -0.000000000704934 \\ 0.000000006778456 \\ -0.000000030168694 \\ 0.000000070771512 \\ -0.000000090340832 \\ 0.000000059336480 \\ -0.000000015697754 \end{bmatrix}.$$

The 2-norm relative error is about $1.3 \cdot 10^{-7}$. Matlab did not "know" that ■

■ **Example 9.10** Here we'll take a detailed look at the Frobenius norm condition number $\|\mathbf{A}\|_F \|\mathbf{A}^{-1}\|_F$ for a 2×2 matrix $\mathbf{A}$. Why the Frobenius norm? Because we can compute it easily by hand, that's why. If we are given

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \qquad \|\mathbf{A}\|_F = \sqrt{a^2+b^2+c^2+d^2},$$

then the standard formula for 2×2 matrix inversion gives

$$\mathbf{A}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}, \qquad \|\mathbf{A}^{-1}\|_F = \frac{1}{|ad-bc|}\sqrt{a^2+b^2+c^2+d^2}.$$

Thus, $\kappa_F(\mathbf{A}) = \frac{a^2+b^2+c^2+d^2}{|ad-bc|}$. Two circumstances where this number could be large are

- if there is significant cancellation in the determinant $ad - bc$, or
- one column (or row) has significantly larger norm than the other.

The condition "cancellation in the determinant" roughly means that the two columns (or rows) are very close to parallel. So a well-conditioned matrix should have columns of roughly equal length, which are preferably close to orthogonal. More generally, matrices that are close to orthogonal are well-conditioned, and matrices that are nearly low-rank are poorly conditioned. ■

In the next chapter, we'll introduce a tool for determining the 2-norm condition number for general matrices, as well as for solving the rank-approximation problem.

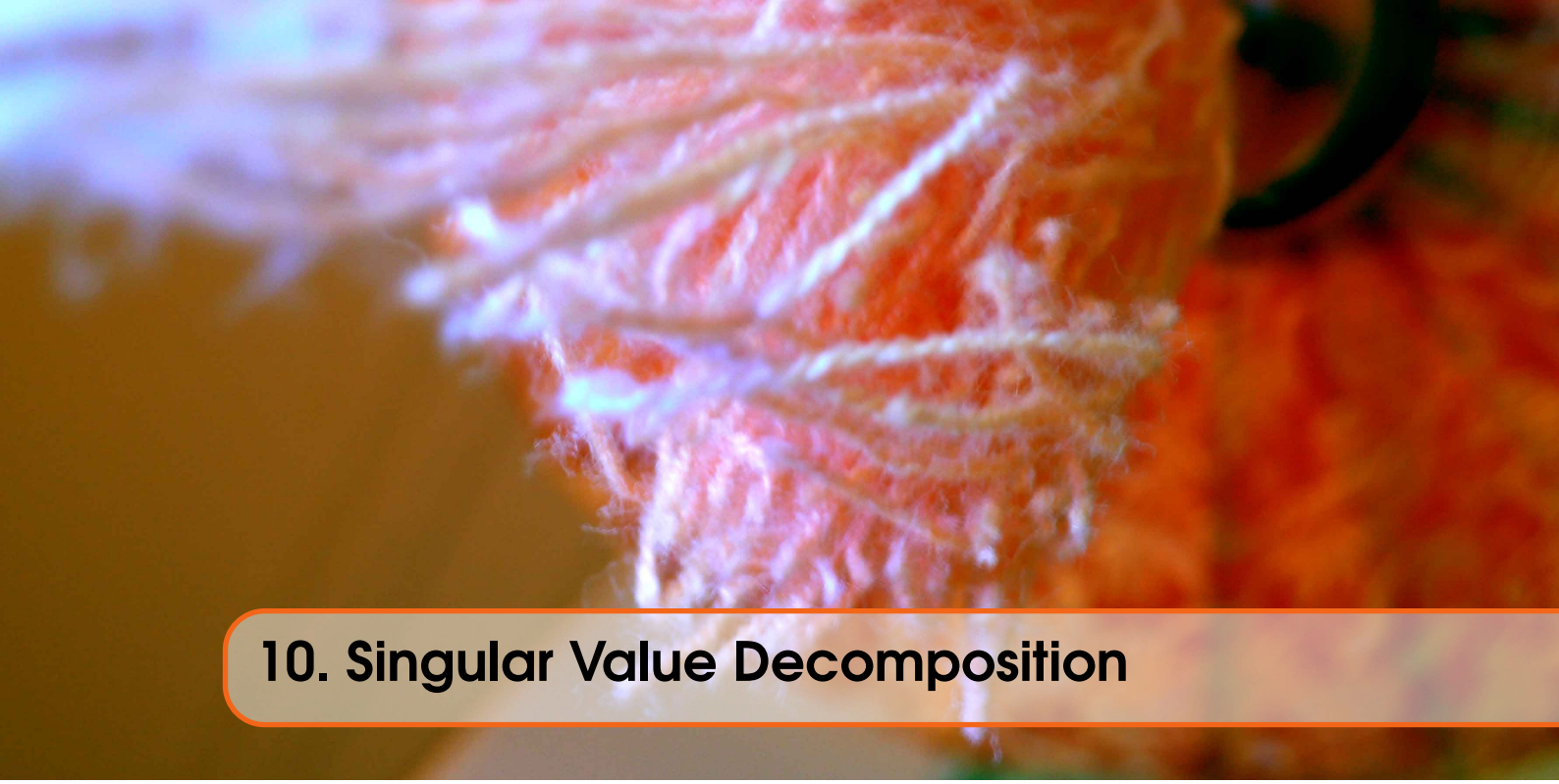
9.4 Matlab Commands

Condition Numbers

- `cond(A)`: the 2-norm condition number of `A`. Using `cond(A,P)` for `P = 1,2,inf,'fro'` gives the condition number in other norms, defined more generally as $\|\mathbf{A}\|\|\mathbf{A}^{-1}\|$.
- `condest(A)`: estimator for the 1-norm condition number of `A`. Useful when `A` is too large to make a more exact estimate practical.
- `rcond(A)`: estimator for the inverse of the 1-norm condition number of `A`.

Test Matrices

- `gallery`: a collection of test matrices. For example, `gallery('cauchy',n)` gives the $n \times n$ Cauchy matrix.



10. Singular Value Decomposition

It's time to tie everything together with one of the most powerful tools in linear algebra.

10.1 Basic Properties

Theorem 10.1.1 — Singular Value Decomposition. For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$, there exists a factorization $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{m\times n}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ have orthonormal columns and $\Sigma \in \mathbb{R}^{n\times n}$ is a diagonal matrix with diagonal elements $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$. The matrix $\mathbf{U}$ can be extended to an orthonormal basis $[\mathbf{U}, \overline{\mathbf{U}}]$ for $\mathbb{R}^m$, in which case

$$\mathbf{A} = [\mathbf{U}, \overline{\mathbf{U}}] \begin{bmatrix} \Sigma \\ \mathbf{0}_{(m-n)\times n} \end{bmatrix} \mathbf{V}^T. \tag{10.1}$$

An analogous factorization exists for matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m > n$.

Definition 10.1.1 The following terminology is used with respect to Theorem 10.1.1:

- The factorization $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ is called the *economy-size* or *thin* SVD of $\mathbf{A}$.
- The factorization in Equation 10.1 is called the *full* SVD of $\mathbf{A}$.
- The columns $\mathbf{u}_1, \ldots, \mathbf{u}_n$ of $\mathbf{U}$ are called *left singular vectors*.
- The columns $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of $\mathbf{V}$ are called *right singular vectors*.
- The diagonal elements $\sigma_1, \ldots, \sigma_n$ are called the *singular values*.

We can define singular value triples in a manner analogous to the way that an eigenpair $(\mathbf{w}, \lambda)$ satisfies $\mathbf{A}\mathbf{w} = \lambda\mathbf{w}$.

Definition 10.1.2 A *singular value triple* is a 3-tuple $(\mathbf{u}, \mathbf{v}, \sigma)$ with $\mathbf{u} \neq \mathbf{0}$, $\mathbf{v} \neq \mathbf{0}$ satisfying $\mathbf{A}\mathbf{v} = \sigma\mathbf{u}$ and $\mathbf{A}^T\mathbf{u} = \sigma\mathbf{v}$.

The singular value decomposition gives a great deal of information about the matrix $\mathbf{A}$, but it comes at a fairly steep price. For comparison, recall that the cost of solving the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is about $\frac{2}{3}n^3$.

Fact 10.1.2 The economy-size SVD of $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ can be computed at a cost of approximately $\min\{14mn^2 + 8n^3,\ 6mn^2 + 20n^3\}$ flops. If only the singular values are required, they can be computed at a cost of approximately $\min\{4mn^2 - 4n^3/3,\ 2mn^2 + 2n^3\}$ flops.

■ **Example 10.1** The matrix $\mathbf{A} = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$ has the singular value decomposition

$$\mathbf{A} = \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{V}^T}.$$

Its singular values are $\sigma_1 = 2\sqrt{2}$ and $\sigma_2 = \sqrt{2}$. ■

10.1.1 Relation to Eigenvalue Decomposition

How do you compute the SVD? One somewhat unhelpful answer is that you don't; the computer does. Except for extremely simple cases, it is impractical to try to find the SVD of a matrix by hand. The specific algorithms used for the SVD are beyond the scope of this text, but are discussed in detail in [23]. Instead, we will stick to presenting the relation between the singular value decomposition and the eigenvalue decomposition.

Theorem 10.1.3 — Spectral Theorem. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Then $\mathbf{A}$ is diagonalizable, all of its eigenvalues are real, and it has an orthonormal set of eigenvectors. In other words, there exist an orthogonal matrix $\mathbf{Q}$ and a diagonal matrix Λ so that $\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^T$.

When $\mathbf{A}$ is symmetric with all nonnegative eigenvalues, the SVD and eigendecomposition coincide: $\mathbf{U} = \mathbf{V} = \mathbf{Q}$ and $\Sigma = \Lambda$. If $\mathbf{A}$ is symmetric but has some negative eigenvalues, then the singular values of $\mathbf{A}$ are equal to the absolute values of the eigenvalues.

From the SVD of $\mathbf{A}$ it is straightforward to verify the following identities.

Fact 10.1.4 If the SVD of $\mathbf{A}$ is $\mathbf{U}\Sigma\mathbf{V}^T$, then $\mathbf{A}^T\mathbf{A} = \mathbf{V}\Sigma^2\mathbf{V}^T$ and $\mathbf{A}\mathbf{A}^T = \mathbf{U}\Sigma^2\mathbf{U}^T$.

Consequently, the singular values of $\mathbf{A}$ are closely related to the eigenvalues of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$.

Fact 10.1.5 The singular values of $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ satisfy the identity

$$\sigma_i(\mathbf{A}) = \sqrt{\lambda_i(\mathbf{A}^T\mathbf{A})} = \sqrt{\lambda_i(\mathbf{A}\mathbf{A}^T)}, \quad 1 \leq i \leq n,$$

where the eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ are assumed to be in descending order.

In practice, finding the eigenvalues of $\mathbf{A}^T\mathbf{A}$ is a less-than-ideal method for computing the singular values of $\mathbf{A}$ due to the effects of rounding error on the smaller singular values and corresponding vectors. If only the largest singular values of $\mathbf{A}$ are required, however, it is fairly efficient.

10.2 Rank and Subspace

From the singular value decomposition, a rank factorization of $\mathbf{A}$ can be immediately obtained.

Definition 10.2.1 Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ have singular values $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$. The *compact* SVD of $\mathbf{A}$ is the factorization $\mathbf{A} = \mathbf{U}_r\Sigma_r\mathbf{V}_r^T$, where

$$\mathbf{U}_r = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \in \mathbb{R}^{m\times r}, \quad \Sigma_r = \operatorname{diag}(\sigma_1 \ldots, \sigma_r) \in \mathbb{R}^{r\times r}, \quad \mathbf{V}_r = [\mathbf{v}_1, \ldots, \mathbf{v}_r] \in \mathbb{R}^{n\times r}.$$

Now $\mathbf{U}_r$ and $\mathbf{V}_r$ both have rank r since they both have orthonormal columns, and Σ_r has rank r since it is a diagonal $r \times r$ matrix with nonzero elements on the diagonal. From Theorem 9.1.2 it follows that $\mathbf{A}$ has rank r.

Theorem 10.2.1 The rank of $\mathbf{A}$ is equal to the number of nonzero singular values of $\mathbf{A}$.

The SVD also gives us orthonormal bases for the fundamental subspaces of $\mathbf{A}$ (Definition 8.3.4).

Theorem 10.2.2 Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ have rank r and the compact SVD $\mathbf{U}_r\Sigma_r\mathbf{V}_r^T$. Extend $\mathbf{U}_r$ and $\mathbf{V}_r$ to orthonormal bases $[\mathbf{U}_r, \mathbf{U}_\perp]$ and $[\mathbf{V}_r, \mathbf{V}_\perp]$ for $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively. Then

- The columns of $\mathbf{U}_r$ form an orthonormal basis for $\mathscr{R}(\mathbf{A})$.
- The columns of $\mathbf{V}_r$ form an orthonormal basis for $\mathscr{R}(\mathbf{A}^T)$.
- The columns of $\mathbf{V}_\perp$ form an orthonormal basis for $\mathscr{N}(\mathbf{A})$.
- The columns of $\mathbf{U}_\perp$ form an orthonormal basis for $\mathscr{N}(\mathbf{A}^T)$.

In theory, the SVD can be used to solve systems of linear equations.

Fact 10.2.3 Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be invertible and have the SVD $\mathbf{U}\Sigma\mathbf{V}^T$. Then the solution to the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is $\mathbf{x} = \mathbf{V}\Sigma^{-1}\mathbf{U}^T\mathbf{b}$.

In practice, this is not a very efficient strategy. Gaussian elimination uses about $\frac{2}{3}n^3$ flops to solve the linear system, but computing the SVD (by an algorithm given in [23]) requires about $21n^3$ flops.

10.3 Optimal Low-Rank Approximation

The 2-norm and Frobenius norm of $\mathbf{A}$ can be stated simply in terms of the singular values of $\mathbf{A}$.

Theorem 10.3.1 Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ have the economy-size SVD $\mathbf{U}\Sigma\mathbf{V}^T$. Then

- $\|\mathbf{A}\|_2 = \sigma_1$,
- $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^n \sigma_i^2}$.

Proof. Since $\mathbf{U}$ and $\mathbf{V}$ have orthonormal columns, it follows by Fact 8.3.15 that

$$\|\mathbf{A}\|_2 = \|\mathbf{U}\Sigma\mathbf{V}^T\|_2 = \|\Sigma\|_2$$

and

$$\|\mathbf{A}\|_F = \|\mathbf{U}\Sigma\mathbf{V}^T\|_F = \|\Sigma\|_F.$$

For the Frobenius norm, it is straightforward to check that $\|\Sigma\|_F = \sqrt{\sum_{i=1}^n \sigma_i^2}$. As for the 2-norm, we observe that for any $\mathbf{x} \in \mathbb{R}^n$,

$$\|\Sigma\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n (\sigma_i x_i)^2} \leq \sigma_1 \sqrt{\sum_{i=1}^n x_i^2} = \sigma_1\|\mathbf{x}\|_2,$$

and so $\|\Sigma\|_2 \leq \sigma_1$. The reverse inequality $\|\Sigma\|_2 \geq \sigma_1$ is proved by using the vector $\mathbf{x} = \mathbf{e}_1$, and therefore $\|\Sigma\|_2 = \sigma_1$. ■

Perhaps the most important property of all: with the SVD, we can solve the low-rank approximation exactly.

Theorem 10.3.2 — Eckart-Young Theorem. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ ($m \geq n$) have the SVD $\mathbf{U}\Sigma\mathbf{V}^T$, and fix a target rank $k < n$. Define the matrices

$$\mathbf{U}_k = [\mathbf{u}_1, \ldots, \mathbf{u}_k], \quad \mathbf{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k], \quad \Sigma_k = \operatorname{diag}(\sigma_1, \ldots, \sigma_k).$$

Then the matrix $\widetilde{\mathbf{A}}_k = \mathbf{U}_k\Sigma_k\mathbf{V}_k^T = \sum_{i=1}^k \sigma_i\mathbf{u}_i\mathbf{v}_i^T$ is an optimal rank-k approximation to $\mathbf{A}$ for both the 2-norm and Frobenius norm, satisfying

$$\|\widetilde{\mathbf{A}}_k - \mathbf{A}\|_2 = \sigma_{k+1} \quad \text{and} \quad \|\widetilde{\mathbf{A}}_k - \mathbf{A}\|_F = \left\| \sum_{i=k+1}^{n} \sigma_i\mathbf{u}_i\mathbf{v}_i^T \right\|_F = \sqrt{\sum_{i=k+1}^{n} \sigma_i^2}.$$

The matrix $\widetilde{\mathbf{A}}_k$ is called the *rank-k truncated SVD*.

If we are fortunate, the matrix $\mathbf{A}$ that we are attempting to approximate might have a very clear *singular value gap*, or *spectral gap*. Matrices with obvious spectral gaps occur naturally in some physics applications, but with many data matrices such as joke ratings we are not likely to get so lucky.

A more complicated, but quite useful variant involves finding an approximation $\mathbf{A} \approx \widetilde{\mathbf{A}}_k + \mathbf{E}$, where $\widetilde{\mathbf{A}}_k$ is low-rank and $\mathbf{E}$ is *sparse* (most of its entries are zero) rather than just having small 2-norm or F-norm. One application of this is in motion detection: a video can be interpreted as a sequence of images over time. If each $m \times n$ image is interpreted as a single vector in $\mathbb{R}^{mn}$, then the video can be interpreted as an $mn \times t$ matrix, where t is the number of time steps. If the background is largely static, then it can be closely approximated by a low-rank matrix. The nonzero components of $\mathbf{E}$ then give an indication as to what the moving objects are in the image. Furthermore, this allows us to store the video for much less memory than would otherwise have been required.

■ **Example 10.2** Figure 10.1 shows an example with the MIT logo. The image matrix $\mathbf{A}$ has a clear singular value gap at rank $k = 5$, with $\sigma_5/\sigma_6 \approx 60$. Consequently, the rank 5 approximation is of fairly high quality, and the rank 6 approximation is not visibly better. ■

■ **Example 10.3** Figure 10.2 shows three stills from a video game in which a character moves across a static background. We might be interested in separating the character from the background, either in order to compress the video effectively or to train an algorithm to identify moving objects. Figure 10.3 shows that just taking the rank-1 truncated SVD does not quite give us the background that we are looking for, since the low-rank approximation still contains some vestiges of the character sprite. A different algorithm is therefore required. ■

10.3.1 Distance to Singularity

Armed with the Eckart-Young Theorem, we can now define precisely what it means for an invertible matrix to *almost* be non-invertible. All we have to do for an $n \times n$ matrix is find an optimal rank-$(n-1)$ approximation, and consider it as a perturbed version of the original matrix.

Theorem 10.3.3 For an invertible matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, the smallest perturbation to $\mathbf{A}$ that would make it singular satisfies

$$\|\mathbf{E}\|_2 = \|\mathbf{E}\|_F = \sigma_{\min}(\mathbf{A}).$$

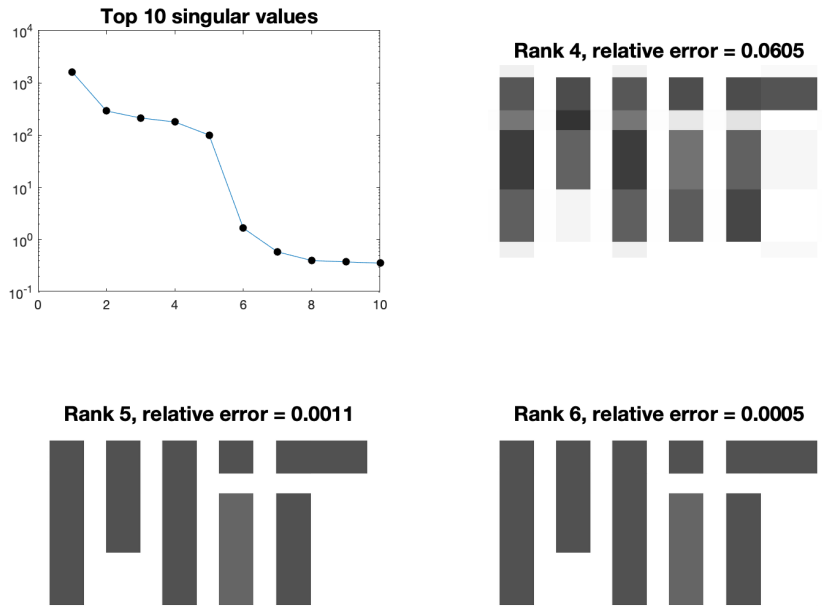


Figure 10.1: Rank 4, 5, and 6 approximations of image with MIT logo.

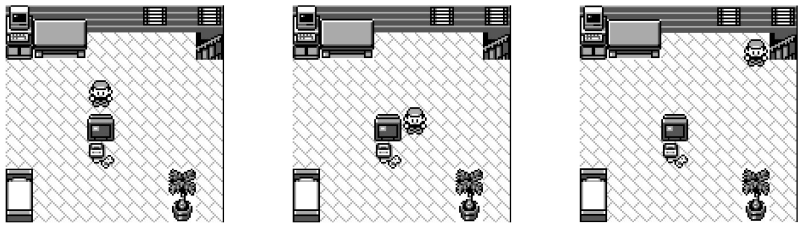


Figure 10.2: Three stills from a video game where the character sprite moves across a static background.

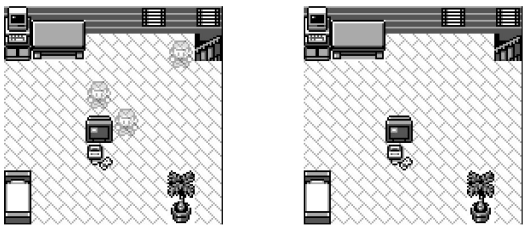


Figure 10.3: Left: rank-1 truncated SVD does not quite give the desired background. Right: ideal background.

One such perturbation is $\mathbf{E} = -\sigma_n \mathbf{u}_n \mathbf{v}_n^T$, and it is unique if and only if $\sigma_{n-1} > \sigma_n$.

Proof. By the Eckart-Young Theorem, an optimal approximation $\widetilde{\mathbf{A}}_{(n-1)}$ satisfies

$$\|\widetilde{\mathbf{A}}_{(n-1)} - \mathbf{A}\|_2 = \|\widetilde{\mathbf{A}}_{(n-1)} - \mathbf{A}\|_F = \sigma_{\min}(\mathbf{A}).$$

■

In terms of *relative* perturbations, this means that a perturbation as small as $\|\mathbf{E}\|_2/\|\mathbf{A}\|_2 = \sigma_n/\sigma_1$ could cause the matrix to become singular.

■ **Example 10.4** The matrix from example 9.8,

$$\mathbf{A} = \begin{bmatrix} 1+10^{-4} & 1 \\ 1 & 1+10^{-4} \end{bmatrix},$$

has singular values $\sigma_1 = 2+10^{-4}$ and $\sigma_2 = 10^{-4}$. The matrix

$$\mathbf{A}+\mathbf{E} = \begin{bmatrix} 1+10^{-4} & 1+10^{-4} \\ 1+10^{-4} & 1+10^{-4} \end{bmatrix}$$

is singular, and satisfies $\|\mathbf{E}\|_2/\|\mathbf{A}\|_2 = \sigma_2/\sigma_1$. The greater the ratio between the largest and smallest singular values, the closer in a relative sense the matrix is to singularity. ■

We will see in Section 10.4.1 that $\sigma_2/\sigma_1 = 1/\kappa(\mathbf{A})$ (the reciprocal of the condition number). This connection can be summarized in the following statement:

An *ill-conditioned* matrix is one that is, relative to its magnitude, *close to singularity*.

10.3.2 (Non-)Uniqueness of the Optimal Approximation

Although we might occasionally refer to *the* singular value decomposition of a matrix, the decomposition is not quite unique.

■ **Example 10.5** The identity matrix has the SVD $\mathbf{I} = \underbrace{\mathbf{Q}}_{\mathbf{U}} \underbrace{\mathbf{I}}_{\Sigma} \underbrace{\mathbf{Q}^T}_{\mathbf{V}^T}$ for any orthogonal $\mathbf{Q}$. ■

■ **Example 10.6** If $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$, then $\mathbf{A} = (-\mathbf{U})\Sigma(-\mathbf{V})^T$. More generally, for any diagonal matrix $\mathbf{D}$ with diagonal entries ± 1, $\mathbf{A} = (\mathbf{UD})\Sigma(\mathbf{VD})^T$. ■

In this section, we clarify the extent to which the SVD is uniquely defined. First, there is no question about the singular *values* of a matrix.

Fact 10.3.4 For any matrix $\mathbf{A}$, the singular values $\sigma_1 \geq \cdots \geq \sigma_{\min\{m,n\}}$ of $\mathbf{A}$ are uniquely defined.

Rather than requiring that a singular value σ_i be part of a specific triple $(\mathbf{u}_i, \mathbf{v}_i, \sigma_i)$, we can introduce the broader notion of a singular subspace.

Definition 10.3.1 — Singular Subspace. For a matrix $\mathbf{A}$ and singular value σ of $\mathbf{A}$, the singular subspaces $\mathscr{U}_\sigma$ and $\mathscr{V}_\sigma$ associated with σ are the largest subspaces satisfying the following:

- For any $\mathbf{u} \in \mathscr{U}_\sigma$, there exists $\mathbf{v} \in \mathscr{V}_\sigma$ such that $(\mathbf{u}, \mathbf{v}, \sigma)$ is a singular triple of $\mathbf{A}$.
- For any $\mathbf{v} \in \mathscr{V}_\sigma$, there exists $\mathbf{u} \in \mathscr{U}_\sigma$ such that $(\mathbf{u}, \mathbf{v}, \sigma)$ is a singular triple of $\mathbf{A}$,

The key distinction is that if a matrix $\mathbf{A}$ has a repeated singular value $\sigma_{k+1} = \cdots = \sigma_{k+d} = \sigma$, then the associated singular subspaces $\mathscr{U}_\sigma$ and $\mathscr{V}_\sigma$ will be d-dimensional. Armed with this broader definition, we get the desired uniqueness conditions for the SVD.

Theorem 10.3.5 — Uniqueness conditions for SVD. For any matrix $\mathbf{A}$, the singular values σ of $\mathbf{A}$ and associated singular subspaces $\mathscr{U}_\sigma$ and $\mathscr{V}_\sigma$ are uniquely defined. If $\mathbf{U}_\sigma$ and $\mathbf{V}_\sigma$ are matrices whose columns respectively form orthonormal bases for $\mathscr{U}_\sigma$ and $\mathscr{V}_\sigma$, then

$$\mathbf{A} = \sum_\sigma \sigma \mathbf{U}_\sigma \mathbf{V}_\sigma^T.$$

The matrices $\mathbf{U}_\sigma$ and $\mathbf{V}_\sigma$ are unique up to right multiplication by an orthogonal matrix: for any orthogonal $\mathbf{Q}$, the pair $(\mathbf{U}_\sigma\mathbf{Q}, \mathbf{V}_\sigma\mathbf{Q})$ would work equally well.

For the final part of the theorem, note in particular that for any (appropriately-sized) orthogonal matrix $\mathbf{Q}$, we would have

$$\sigma(\mathbf{U}_\sigma\mathbf{Q})(\mathbf{V}_\sigma\mathbf{Q})^T = \sigma\mathbf{U}_\sigma\mathbf{Q}\mathbf{Q}^T\mathbf{V}_\sigma^T = \sigma\mathbf{U}_\sigma\mathbf{I}^T\mathbf{V}_\sigma^T = \sigma\mathbf{U}_\sigma\mathbf{V}_\sigma^T.$$

This implies that the value of the matrix product is not affected by the choice of $\mathbf{Q}$. Consequently, we get a theorem about the uniqueness of the optimal rank-k approximation.

Theorem 10.3.6 — Uniqueness conditions for truncated SVD. The optimal (2-norm or F-norm) rank-k approximation $\widetilde{\mathbf{A}}_k$ to a given matrix $\mathbf{A}$ is unique if and only if $\sigma_k > \sigma_{k+1}$.

The reasoning is that the subspaces associated with "the largest k singular values" are uniquely defined only if it is clear which k singular values are the largest. If the kth singular value is larger than the $(k+1)$st, then this will be the case.

In finite precision arithmetic, it is rare for two computed singular values to be precisely equal to one another unless the matrix in question has some clear structure. Instead, if two singular values are very close to one another then we might say that the associated singular vectors are highly sensitive to perturbations in the data.

■ **Example 10.7** The matrix

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

has singular values $\sigma_1 = 3$, $\sigma_2 = \sigma_3 = 2$. I therefore has a unique optimal rank-1 approximation, but infinitely many optimal rank-2 approximations.

$$\widetilde{\mathbf{A}}_1 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \widetilde{\mathbf{A}}_2 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & x^2 & x\sqrt{2-x^2} \\ 0 & x\sqrt{2-x^2} & 2-x^2 \end{bmatrix}, \quad x \in [-\sqrt{2}, \sqrt{2}].$$

■

10.4 SVD as a Linear Transformation

Time to take a break from approximations for a while, and focus on what the SVD means in terms of linear transformations. Interpreting matrix factorizations as function composition, the factorization $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ represents an *orthogonal transformation*, then a *rescaling* along the coordinate axes, followed by another *orthogonal transformation*.

We complement this interpretation with a theorem closely related to the Principal Axis Theorem, which shows how to find the principal axes of an ellipsoid or hyperboloid.

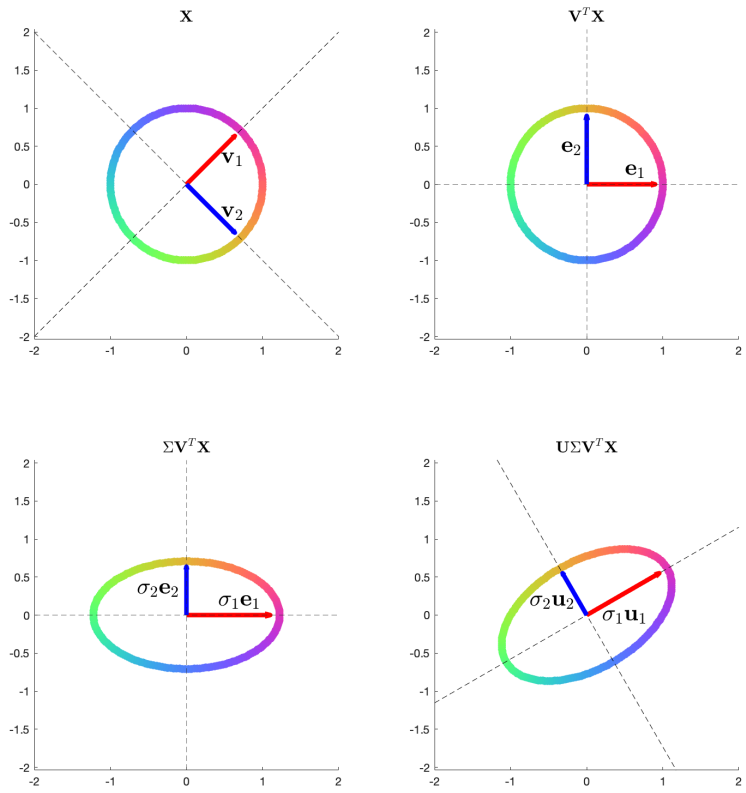


Figure 10.4: A linear transformation takes the unit sphere to an ellipsoid. Pictures show the effects of the SVD factors applied one at a time. Final semiaxis lengths are equal to the singular values. Columns of $\mathbf{X} \in \mathbb{R}^{2\times n}$ represent points on the unit circle.

Theorem 10.4.1 — Principal Axis Theorem. For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the set of points $\{\mathbf{Ax} \ : \ \|\mathbf{x}\|_2 = 1\}$ defines an ellipsoid in $\mathbb{R}^m$ with dimensions equal to the rank of $\mathbf{A}$. The principal axes of the ellipsoid lie in the directions of the left singular vectors of $\mathbf{A}$, and the semiaxis lengths are equal to the corresponding singular values of $\mathbf{A}$.

Put more broadly: a linear transformation does not just map lines to lines, it also maps ellipsoids to ellipsoids.

■ **Example 10.8** Consider the matrix

$$\mathbf{A} = \begin{bmatrix} \frac{1}{2} & 1 \\ \frac{\sqrt{3}}{2} & 0 \end{bmatrix},$$

with singular values $\sigma_1 = \sqrt{3/2}$ and $\sigma_2 = \sqrt{1/2}$. Figure 10.4 shows that applying $\mathbf{V}^T$ has the effect of a rotation-plus-reflection. Multiplying by Σ scales the coordinate axes by the corresponding singular values, and $\mathbf{U}$ finally applies another rotation. In all, the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are respectively mapped to $\sigma_1\mathbf{u}_1$ and $\sigma_2\mathbf{u}_2$. ■

10.4.1 Relation to Condition Number

We may accompany Theorem 10.4.1 with a statement about how much a linear transformation $\mathbf{A}$ may distort the length of a vector.

Theorem 10.4.2 For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ with singular values $\sigma_1 \geq \cdots \geq \sigma_n$, the following identities hold:

- $\max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2 = \sigma_1$,
- $\min_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2 = \sigma_n$.

With respect to Figure 10.4, the largest and smallest singular values correspond to the greatest and least amounts by which the linear transformation $\mathbf{A}$ scales the length of a vector. Turning back to the notion of *aspect ratio* discussed with Fact 9.3.3, we draw the following conclusion.

Theorem 10.4.3 For a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, the condition number is given by $\kappa(\mathbf{A}) = \sigma_1/\sigma_n$.

This theorem can also be deduced from the definition of $\kappa(\mathbf{A})$ (Definition 9.3.1):

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|_2\|\mathbf{A}^{-1}\|_2 = \sigma_1/\sigma_n,$$

since $\mathbf{A}^{-1}$ has singular values $\sigma_n^{-1} \geq \cdots \geq \sigma_1^{-1}$. The professor somewhat prefers to think about the condition number in terms of its connection to Theorem 10.4.1, since this theorem provides some visual intuition: the condition number of a matrix is the greatest amount by which it may distort a ratio of two lengths.

The worst-case scenario, in terms of the sensitivity of the system $\mathbf{A}\mathbf{x} = \mathbf{b}$, is when $\mathbf{b}$ is aligned with $\mathbf{u}_1$, or generally the left singular vectors associated with the largest singular values of $\mathbf{A}$, and $\Delta\mathbf{b} = \tilde{\mathbf{b}} - \mathbf{b}$ is aligned with $\mathbf{u}_n$ or the other left singular vectors associated with the smallest singular values of $\mathbf{A}$.

10.5 Practical Computation

As was previously mentioned, the singular value decomposition can be rather expensive to compute. The cost is $\mathcal{O}(mn^2)$ for an $m \times n$ matrix with $m \geq n$, and when the matrix is large this cost can be prohibitively expensive. It is also wasteful if we only need a small number of the singular values of $\mathbf{A}$ (say, the k largest or smallest) or a rank-k approximation to $\mathbf{A}$ with $k \ll \min\{m,n\}$. Depending on the situation, specialized algorithms exist that can be more efficient than the most general ones.

- If $\mathbf{A}$ is tall-skinny $(m > n)$, then computing the economy-size SVD (`econ` in Matlab) is more efficient than computing the full SVD.
- If only the singular values are desired, or only one of $\mathbf{U}$ or $\mathbf{V}$ rather than both, algorithms exist that are even more efficient. In Matlab, `s = svd(A)` computes only the singular values.
- If only a few singular values are required (e.g., the largest k, smallest k, smallest k nonzero values, or the k values closest to a given number σ), then `svds(A,k)` can be more efficient. The underlying algorithm uses $\mathbf{A}$ only for matrix-vector multiplications $\mathbf{A}\mathbf{x}$ and $\mathbf{A}^T\mathbf{y}$, so it is particularly efficient when $\mathbf{A}$ is stored in a sparse format.

If $\mathbf{A}$ has rank exactly equal to r, then in exact arithmetic Gaussian elimination would terminate after r steps and return a rank factorization for $\mathbf{A}$. The total cost would be $\mathcal{O}(mnr)$ flops, which is significantly smaller than mn^2 when $r \ll \min\{m,n\}$. In theory, we could use a rank factorization $\mathbf{A} = \mathbf{L}\mathbf{R}$ to find the economy-size SVD of $\mathbf{A}$ at a cost of $\mathcal{O}((m+n)r^2)$ additional flops, as shown in

Program 10.1: SVD from a rank factorization

```
% Inputs: Factors L (m x r), R (r x n) such that A = LR
% Outputs: Factors U,S,V for SVD of A
[U,S,V] = svd(L,'econ'); % cost: O(mr^2)
R = S*V'*R; % cost: O(nr^2)
[Q,S,V] = svd(R,'econ') % cost: O(nr^2)
U = U*Q; % cost: O(mr^2)
```

Program 10.2: Randomized matrix sketching

```
% Inputs: Matrix A (m x n), rank r
% Outputs: Factors U,S,V for approximate SVD of A
Omg = randn(n,r); % randomized sketching matrix
Z = A*Omg; % the sketch B is (m x r)
Q = orth(Z); % find an orthonormal basis for R(B)
B = Q'*A; % A is approximately (Q*Q')A = Q*B
[U,S,V] = svd(B,'econ');
U = Q*U; % approximate SVD for A
```

Program 10.1. If $\mathbf{A}$ is exactly low rank, it is therefore wasteful to spend $\mathcal{O}(mn^2)$ flops computing the SVD.

If $\mathbf{A}$ can be *well-approximated* by a matrix of rank r, it is often more practical to find a sub-optimal rank-r approximation than to find the SVD and take its rank-r truncation. We can hope to find approximations that are *nearly* as good as the bounds given by the Eckart-Young theorem, but at a cost of $\mathcal{O}(mnr)$ rather than $\mathcal{O}(mn^2)$. Some algorithms designed for this task are commonly known as *rank-revealing* algorithms: for example, one might run a variant of Gaussian elimination that makes strategic choices for the row (and column) pivoting and terminates after r steps. Another class uses a technique known as *sketching*, which roughly involves mapping $\mathbf{A}$ down to a smaller-dimensional space and examining the output to learn about $\mathbf{A}$. Program 10.2 shows a basic version of a *randomized* sketching algorithm that gives a low-rank approximation to $\mathbf{A}$. The key idea is that when $\Omega \in \mathbb{R}^{n\times r}$ is a matrix with randomly drawn entries, the column space of $\mathbf{A}\Omega$ will (with high probability) contain a good approximation to the top singular subspace of $\mathbf{A}$.

- The Matlab function `svdsketch` uses randomized sketching to cheaply find a low-rank approximation to $\mathbf{A}$.

10.6 Matlab Commands

SVD tools

- `s = svd(A)`: Returns a vector with the singular values of `A`.
- `s = svds(A)`: More efficient if only a few singular values of `A` are required. See the documentation for more information.
- `[U,S,V] = svd(A)`: Computes the full singular value decomposition of `A`. For the "economy-size" decomposition, use `svd(A,0)`.
- `[U,S,V] = svdsketch(A,tol)`: Uses a randomized algorithm to compute a low-rank approximation to `A` with a relative Frobenius norm error bounded by `tol`.
- `norm(A)`: Returns the 2-norm of `A`, equal to the largest singular value of `A`. Use `norm(A,1)`, `norm(A,Inf)`, `norm(A,'fro')` for the 1-norm, infinity-norm, and Frobenius norm.

Sparse matrices

- `sparse(A)`: Convert a matrix `A` to sparse format.
- `sparse(m,n)`: Creates the $m \times n$ sparse all-zero matrix.
- `full(A)`: Convert a sparse matrix to regular format.
- `spy(A)`: Make a plot showing the location of the nonzero entries in `A`.
- `speye(n)`: The sparse $n \times n$ identity matrix.
- `spdiags(B,d,m,n)`: creates an $m \times n$ sparse matrix from the columns of `B` and places them along the diagonals specified by `d`.

11. Projections and PCA

Why is it so valuable to have an orthonormal basis for a given subspace? One reason is that they allow us to cheaply *project* a given vector onto a subspace. What's so important about that? That's the topic for this chapter!

11.1 Three Types of Projection

In graphics or technical drawing, the term *projection* is commonly used to mean a method for displaying a three-dimensional object on a two-dimensional surface. It comes in several varieties, which we discuss here.

11.1.1 Perspective Projections

From a computer graphics perspective, the most natural type of projection is a *perspective projection*. Designed to mimic the way we perceive objects in the real world, it can be described in terms of a *camera* (i.e. a viewer's eye) and the *plane* onto which the underlying object will be projected. The *lines of projection* are straight lines from the object to the camera.

■ **Example 11.1** Figure illustrates a perspective projection where the camera is located at the origin and the plane is described by the equation $y = 1$, or the set of points $\{(x,1,z) : x,z \in \mathbb{R}\}$. Points in $\mathbb{R}^3$ are mapped to the plane via the transformation $(x,y,z) \mapsto (x/y, 1, z/y)$, which can be represented in two dimensions as the mapping $(x,y,z) \mapsto (x/y, z/y)$. ■

It is important to note that a perspective projection is *not* a linear transformation. Looking at Example 11.1, we can check that the function $f : \mathbb{R}^3 \to \mathbb{R}^2$ given by $f(x,y,z) = (x/y, z/y)$ satisfies $f(\alpha x, \alpha y, \alpha z) = f(x,y,z)$ for any nonzero $\alpha \in \mathbb{R}$. The function f therefore does not satisfy the linearity property $f(\alpha \mathbf{x}) = \alpha f(\mathbf{x})$. Visually, we note that the lines of projection are not parallel to one another. Consequently, lines that were parallel in the underlying space may not remain parallel once they are projected: in Figure 11.1, the squares appear as trapezoids. Because the focus of this section is on linear algebra techniques, we will not consider perspective projections further.

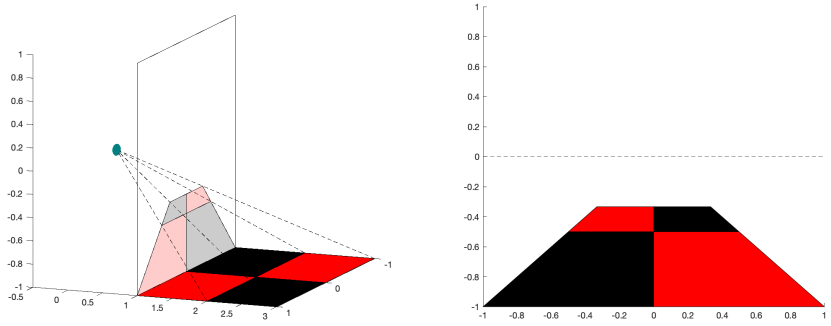


Figure 11.1: Left: Illustration of perspective projection mapping. Right: Perceived image.

11.1.2 Oblique Projections

The most important type of projection aside from a perspective projection is a *parallel projection*, one in which the lines of projection from the object to the plane are parallel. This does not mimic human vision as closely, but it does have the attractive property of mapping parallel lines to parallel lines (useful for technical drawings), and it can be represented by a linear transformation. It can also be viewed as the limit of a perspective projection as the camera moves away from the plane: for example, from the point of view of a fixed spot on Earth, the light rays coming from the sun are nearly parallel at any given point in time. If the lines of projection are themselves normal to the target plane, the projection is called an *orthogonal* projection, or *orthographic* in the case of projecting from three dimensions to two dimensions. If the lines of projection are not normal to the target plane, the projection is called an *oblique* or *skew* projection.

■ **Example 11.2** The transformation shown in Figure 11.2, simulating a shadow cast by the sun, is given by the matrix

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

This matrix projects $\mathbb{R}^3$ onto the xy-plane. ■

■ **Example 11.3** The projection onto the y-axis shown in Figure 11.2 is given by the matrix

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

Note that the points $(1,0)$ and $(2,0)$ initially have the same y-coordinate, but are mapped to different points on the y-axis. Meanwhile, $(1,2)$ and $(2,0)$ are mapped to the same point $(0,4)$ on the y-axis. ■

11.1.3 Orthogonal Projections

The type of projection that will be of greatest interest to us is the orthogonal projection: one in which the lines of projection are normal to the target plane.

■ **Example 11.4** Figure 11.3 shows the orthographic projection of a cube onto the xy-plane, given by the matrix

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

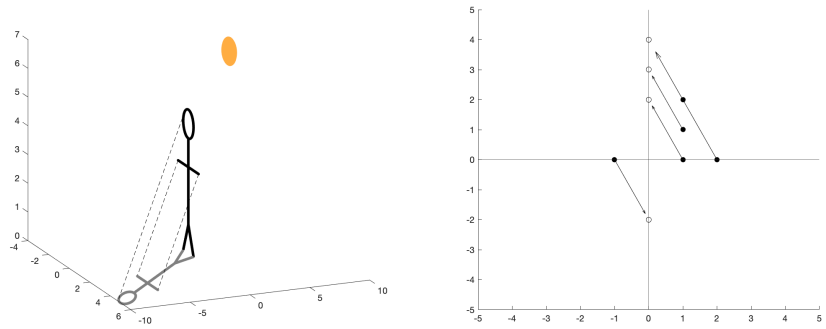


Figure 11.2: Left: shadows cast by the sun can be modeled by an oblique projection (or orthographic if the sun is directly overhead). Lines of projection are parallel to one another. Right: oblique projection onto the y-axis.

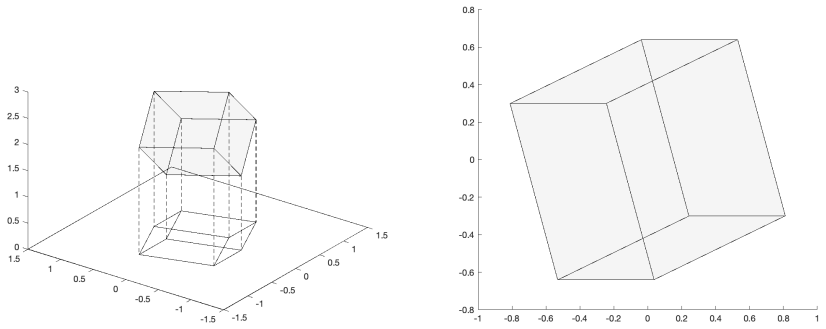


Figure 11.3: Left: orthographic projection of a cube onto the xy-plane. Right: view of the cube from directly overhead.

■

■ **Example 11.5** As shown in Figure 11.4, the matrix

$$\Pi = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

is an orthogonal projection from $\mathbb{R}^2$ to the line defined by the equation $y = x$. ■

■ **Example 11.6** The zero matrix $\mathbf{0}_{n\times n}$ and identity matrix $\mathbf{I}_n$ are both projections, but not particularly interesting ones. ■

11.2 Mathematical Properties

In more formal linear algebra terms, we define projection matrices in the following manner.

Definition 11.2.1 A *projection matrix* $\Pi \in \mathbb{R}^{n\times n}$ is one satisfying $\Pi^2 = \Pi$. The projection is called *orthogonal* if Π is symmetric, and *oblique* if it is not. In statistical settings, Π is sometimes called a *hat matrix*. In a slight abuse of terminology, we will refer to orthogonal projections as *orthographic* projections in order to avoid confusing them with orthgonal matrices.

One interpretation of the property $\Pi^2 = \Pi$ is that for any $\mathbf{x} \in \mathbb{R}^n$, $\Pi(\Pi\mathbf{x}) = \Pi\mathbf{x}$. If a point has been projected once, applying the projection again will not alter it further since it already lies in the

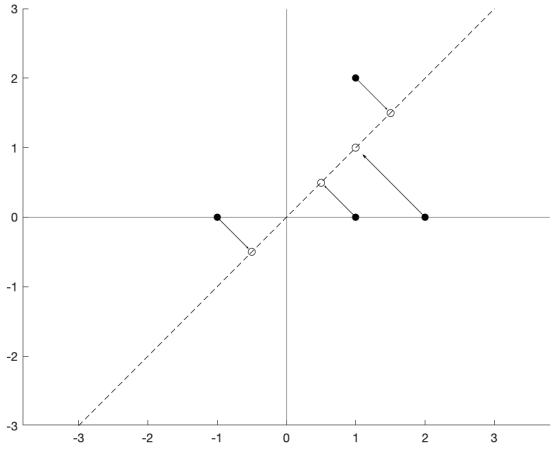


Figure 11.4: Orthogonal projection onto the line $y = x$.

target plane.

Fact 11.2.1 Every projection matrix Π is diagonalizable, and all of its eigenvalues are equal to 0 or 1.

Fact 11.2.2 For every projection matrix $\Pi \in \mathbb{R}^{n\times n}$, the whole space $\mathbb{R}^n$ can be decomposed as the direct sum $\mathbb{R}^n = \mathscr{R}(\Pi) \oplus \mathscr{N}(\Pi)$.

A helpful way to think about these two facts together is that given a projection matrix $\Pi \in \mathbb{R}^{n\times n}$, every point $\mathbf{x} \in \mathbb{R}^n$ can be uniquely written as the sum

$$\mathbf{x} = \Pi\mathbf{x} + (\mathbf{I} - \Pi)\mathbf{x},$$

where the first part lies inside the target subspace $\mathscr{R}(\Pi)$ and the second part lies inside $\mathscr{N}(\Pi)$. The dimension of the first space is the rank of Π, which is equal to the multiplicity of the eigenvalue 1 (for every $\mathbf{x} \in \mathscr{R}(\Pi), \Pi\mathbf{x} = \mathbf{x}$). The dimension of the second space is the multiplicity of the eigenvalue 0: note that for every $\mathbf{x} \in \mathbb{R}^n$, we have

$$\Pi[(\mathbf{I} - \Pi)\mathbf{x}] = (\Pi - \Pi^2)\mathbf{x} = \mathbf{0}_{n\times n}\mathbf{x} = \mathbf{0},$$

since $\Pi = \Pi^2$ by definition.

11.2.1 Relation to Singular Value Decomposition

Note that just because all of the *eigenvalues* of a projection are 0 or 1 does not necessarily mean that all of the *singular values* are equal to 0 or 1. The multiplicity of the eigenvalue 0 is equal to the multiplicity of the singular value 0 because both have multiplicity equal to the dimension of $\mathscr{N}(\Pi)$. But in Example 11.3, the oblique projection Π satisfies $\|\Pi\|_2 = \sigma_1(\Pi) = \sqrt{5}$.

For *orthographic* projections, however, Π is symmetric and so the eigenvalues and singular values coincide.

Theorem 11.2.3 An orthographic projection matrix $\Pi \in \mathbb{R}^{n\times n}$ with rank r has the SVD and

eigenvalue decomposition

$$\Pi = \begin{bmatrix}\mathbf{U}_r, \mathbf{U}_\perp\end{bmatrix} \begin{bmatrix}\mathbf{I}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_{n-r}\end{bmatrix} \begin{bmatrix}\mathbf{U}_r, \mathbf{U}_\perp\end{bmatrix}^T.$$

The columns of $\mathbf{U}_r$ form an orthonormal basis for $\mathscr{R}(\Pi)$, the target subspace.

In visual terms: any rank-r orthographic projection takes the unit sphere in n dimensions and flattens it down to an r-dimensional sphere sitting in the subspace $\mathscr{R}(\mathbf{U}_r)$.

Notation 11.1. *Given a subspace $\mathscr{V} \subseteq \mathbb{R}^n$, the matrix $\Pi_\mathscr{V} \in R^{n\times n}$ will denote the orthographic projection onto $\mathscr{V}$.*

Notation 11.2. *Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the matrix $\Pi_\mathscr{Y} \in R^{m\times m}$ will denote the orthographic projection onto $\mathscr{R}(\mathbf{A})$.*

It is important to note that these matrices are uniquely defined, even if there are many possible orthonormal bases for a given subspace. From Theorem 11.2.3 and the uniqueness conditions for the SVD in Theorem 10.3.5, we get the following result.

Theorem 11.2.4 Let $\mathbf{U}$ be any orthonormal basis for a subspace $\mathscr{V} \subseteq \mathbb{R}^n$. Then the projection $\Pi_\mathscr{V}$ is given by

$$\Pi_\mathscr{V} = \mathbf{U}\mathbf{U}^T.$$

This factorization is unique up to an orthogonal transformation: if $\dim(\mathscr{V}) = r$, then any other orthonormal basis $\mathbf{U}'$ for $\mathscr{V}$ has the form $\mathbf{U}' = \mathbf{U}\mathbf{Q}$, where $\mathbf{Q} \in \mathbb{R}^{r\times r}$ is orthogonal.

Corollary 11.2.5 Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$, let $\mathbf{U}$ be any orthonormal basis for $\mathscr{R}(\mathbf{A})$. Then the projection $\Pi_\mathbf{A}$ is given by

$$\Pi_\mathbf{A} = \mathbf{U}\mathbf{U}^T.$$

This factorization is unique up to an orthogonal transformation in the same manner as before.

■ **Example 11.7** Example 11.5 asserted that the matrix

$$\Pi = \frac{1}{2}\begin{bmatrix}1 & 1 \\ 1 & 1\end{bmatrix}$$

was an orthographic projection onto the line $y = x$. Here we show how to construct the projection given the line: the line $y = x$ corresponds to the subspace $\mathscr{R}([1,1]^T)$. An orthonormal basis for this space is

$$\mathbf{u} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 \\ 1\end{bmatrix},$$

from which we get $\Pi = \mathbf{u}\mathbf{u}^T$. ■

11.2.2 Pythagorean Theorem

Carrying on from Fact 11.2.2, we note that the decomposition $\mathbf{x} = \Pi\mathbf{x} + (\mathbf{I} - \Pi)\mathbf{x}$ takes on special significance when Π is an orthographic projection.

Notation 11.3. *Given a vector $\mathbf{x} \in \mathbb{R}^n$ and a projection Π, we will use the notation*

- $\hat{\mathbf{x}} = \Pi\mathbf{x}$,
- $\mathbf{x}_\perp = (\mathbf{I} - \Pi)\mathbf{x} = \mathbf{x} - \hat{\mathbf{x}}$.

Theorem 11.2.6 For any orthographic projection $\Pi \in \mathbb{R}^{n\times n}$ and vector $\mathbf{x} \in \mathbb{R}^n$, $\hat{\mathbf{x}}$ and $\mathbf{x}_\perp$ are orthogonal.

Proof. Using the facts that $\Pi^2 = \Pi$ and $\Pi = \Pi^T$, we find that

$$\hat{\mathbf{x}}^T\mathbf{x}_\perp = (\Pi\mathbf{x})^T(\mathbf{I}-\Pi)\mathbf{x} = \mathbf{x}^T\Pi^T(\mathbf{I}-\Pi)\mathbf{x} = \mathbf{x}^T\Pi(\mathbf{I}-\Pi)\mathbf{x} = \mathbf{x}^T\mathbf{0}_{n\times n}\mathbf{x} = 0.$$

■

Consequently, the Pythagorean theorem makes a reappearance when we are dealing with orthographic projections.

Corollary 11.2.7 For any orthographic projection $\Pi \in \mathbb{R}^{n\times n}$ and vector $\mathbf{x} \in \mathbb{R}^n$, we will have

$$\|\mathbf{x}\|_2^2 = \|\Pi\mathbf{x}\|_2^2 + \|(\mathbf{I}-\Pi)\mathbf{x}\|_2^2.$$

In other words,

$$\|\mathbf{x}\|_2^2 = \|\hat{\mathbf{x}}\|_2^2 + \|\mathbf{x}_\perp\|_2^2.$$

Since the squared Frobenius norm of a matrix is equal to the sum of the squares of the 2-norms of its columns, a similar result holds for matrices.

Corollary 11.2.8 For any orthographic projection $\Pi \in \mathbb{R}^{m\times m}$ and matrix $\mathbf{X} \in \mathbb{R}^n$,

$$\|\mathbf{X}\|_F^2 = \|\Pi\mathbf{X}\|_F^2 + \|(\mathbf{I}-\Pi)\mathbf{X}\|_F^2.$$

The same goes when left-multiplying by a projection, since the squared Frobenius norm of a matrix is also equal to the sum of the squares of the 2-norms of its rows.

Corollary 11.2.9 For any orthographic projection $\Pi \in \mathbb{R}^{n\times n}$ and matrix $\mathbf{X} \in \mathbb{R}^n$,

$$\|\mathbf{X}\|_F^2 = \|\mathbf{X}\Pi\|_F^2 + \|\mathbf{X}(\mathbf{I}-\Pi)\|_F^2.$$

11.2.3 Special Cases

Here we look at two particular cases. The first is when the subspace that we want to project onto is one-dimensional, in which case we can compute the projection in terms of a single inner product. For the second, we look at an alternate method for expressing the projection onto the column space of a matrix with full column rank.

In the case of projection onto the span of a single vector, the following formula can be commonly found:

Fact 11.2.10 For vectors $\mathbf{x} \neq \mathbf{0} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$, the orthographic projection can be computed as $\Pi_\mathbf{x}\mathbf{y} = \mathbf{x}\left(\frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}}\right)$.

Figure 11.5 illustrates this projection. We derive this formula from the perspective of orthonormal bases: the vector $\mathbf{u} = \mathbf{x}/\|\mathbf{x}\|_2$ is a unit vector parallel to $\mathbf{x}$, and therefore is an orthonormal basis for $\text{Span}\{\mathbf{x}\}$. We can therefore express the projection in terms of the rank-1 matrix

$$\Pi_\mathbf{x} = \mathbf{u}\mathbf{u}^T = (\mathbf{x}/\|\mathbf{x}\|_2)(\mathbf{x}/\|\mathbf{x}\|_2)^T = \frac{1}{\|\mathbf{x}\|_2^2}\mathbf{x}\mathbf{x}^T = \mathbf{x}(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T = \frac{\mathbf{x}\mathbf{x}^T}{\mathbf{x}^T\mathbf{x}}.$$

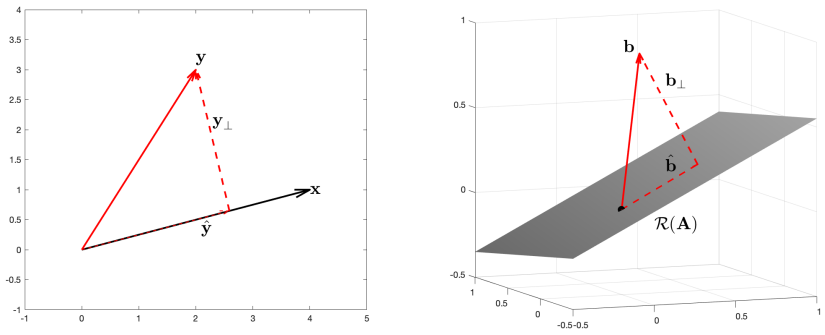


Figure 11.5: Left: projection onto the span of a single vector. Right: projection onto a column space.

Fact 11.2.10 can be derived from the final expression. Note that division by $\mathbf{x}^T\mathbf{x}$ can only be done when $\mathbf{x}$ is a column vector and not for more general matrices.

For the second case, we have the following identity:

Theorem 11.2.11 If a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ has full column rank, then $\mathbf{A}^T\mathbf{A}$ is invertible and

$$\Pi_{\mathbf{A}} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$$

In this context the projection matrix is sometimes called a *hat matrix* because $\Pi_{\mathbf{A}}\mathbf{b} = \hat{\mathbf{b}}$ under Notation 11.3. This expression for the projection matrix can be proved using the economy-size SVD.

Proof. Let $\mathbf{A}$ have the SVD $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$. If $\mathbf{A}$ has full column rank then Σ is invertible, and so

$$(\mathbf{A}^T\mathbf{A})^{-1} = (\mathbf{V}\Sigma^2\mathbf{V}^T)^{-1} = \mathbf{V}\Sigma^{-2}\mathbf{V}^T.$$

It follows that

$$\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \mathbf{U}\Sigma\mathbf{V}^T(\mathbf{V}\Sigma^{-2}\mathbf{V}^T)\mathbf{V}\Sigma\mathbf{U}^T = \mathbf{U}\Sigma\Sigma^{-2}\Sigma\mathbf{U}^T = \mathbf{U}\mathbf{U}^T = \Pi_{\mathbf{A}}.$$

■

Note that this expression for $\Pi_{\mathbf{A}}$ is symmetric and satisfies $\Pi_{\mathbf{A}}^2 = \Pi_{\mathbf{A}}$. This expression can be useful for theoretical purposes, but it is usually a bad idea to form it in practice, particularly if $m \gg n$.

11.3 Dimension Reduction

So where does all of this talk of projections lead us? One of the big questions that we will address is this:

> Given a data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ representing n points in d dimensions, can we find a matrix $\widetilde{\mathbf{X}} \in \mathbb{R}^{n\times k}$ that conveys the same information in $k < d$ dimensions?

This is closely related to the problem of projecting $\mathbf{X}$ onto a k-dimensional subspace. In fact, if $\Pi = \mathbf{Q}\mathbf{Q}^T \in \mathbb{R}^{d\times d}$ is a rank-k projection matrix where $\mathbf{Q} \in \mathbb{R}^{d\times k}$ has orthonormal columns, then the $n \times d$ matrix $\mathbf{X}\Pi = \mathbf{X}\mathbf{Q}\mathbf{Q}^T$ and the smaller $n \times k$ matrix $\mathbf{X}\mathbf{Q}$ contain "essentially the same information" in the sense that there is a natural mapping between the two.

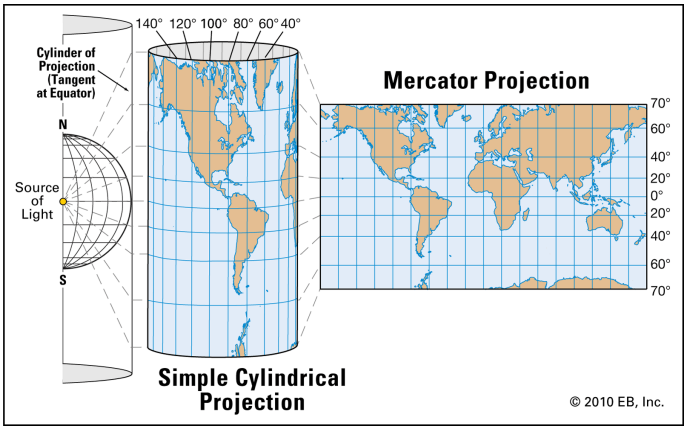


Figure 11.6: The Mercator projection is a nonlinear function from the surface of a sphere to a rectangle. Source: Encyclopedia Britannica [6]

Theorem 11.3.1 Given a k-dimensional subspace $\mathcal{V} \subseteq \mathbb{R}^d$ and matrix $\mathbf{Q} \in \mathbb{R}^{d\times k}$ whose columns form an orthonormal basis for $\mathcal{V}$, the mapping $\mathbf{x} \mapsto \mathbf{Q}^T\mathbf{x}$ is a *linear isometry* from $\mathcal{V}$ to $\mathbb{R}^k$. This means that it preserves 2-norms, and therefore distances and angles as well. The inverse map is given by $\mathbf{y} \mapsto \mathbf{Q}\mathbf{y}$.

So one way to try to answer our original question is to rephrase it in terms of a question about projections,

> Given a data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ representing n points in d dimensions, can we find a k-idmensional subspace $\mathcal{V} \subseteq \mathbb{R}^n$, $k < d$, so that the projection $\mathbf{X}\Pi_{\mathcal{V}} \in \mathbb{R}^{n\times k}$ conveys the same information?

11.3.1 Nonlinear Dimension Reduction

It should be noted that we are implicitly assuming that the map taking our high-dimensional data to a low-dimensional space is a *linear* map: in this case, $\mathbf{X} \mapsto \mathbf{X}\mathbf{Q}$ where $\mathbf{Q}$ has orthonormal columns. In practice, we might be interested in nonlinear dimension reduction techniques as well. One application is in map drawing, where we face the problem of representing the surface of the globe as a flat two-dimensional image. There is no single "correct" way to do this, and various strategies have their advantages and disadvantages. For example, the Mercator projection shown in Figure 11.6 distorts areas (particularly near the poles), but is useful for navigation since straight lines on the map correspond to straight paths for a ship.

As a toy example, consider the data matrix $\mathbf{X} \in \mathbb{R}^{8\times 2}$ with entries

$$x_{j1} = \cos\left(\pi\left(j/9 - 0.5\right)\right), \quad x_{j2} = \sin\left(\pi\left(j/9 - 0.5\right)\right), \quad 1 \le j \le 8.$$

As shown in Figure 11.7, the rows of $\mathbf{X}$ (i.e. the data) all lie on a semicircle of radius 1 centered at the origin. We can parameterize this semicircle in terms of a single variable t by the function

$$f(t) = [\cos(\pi(t-0.5)), \sin(\pi(t-0.5))],$$

which maps the closed interval $[0,1]$ onto the semicircle. We can then represent $\mathbf{X}$ by the smaller matrix

$$\widetilde{\mathbf{X}} = \begin{bmatrix} 1/9 \\ 2/9 \\ \vdots \\ 8/9 \end{bmatrix} \in \mathbb{R}^{8\times 1},$$

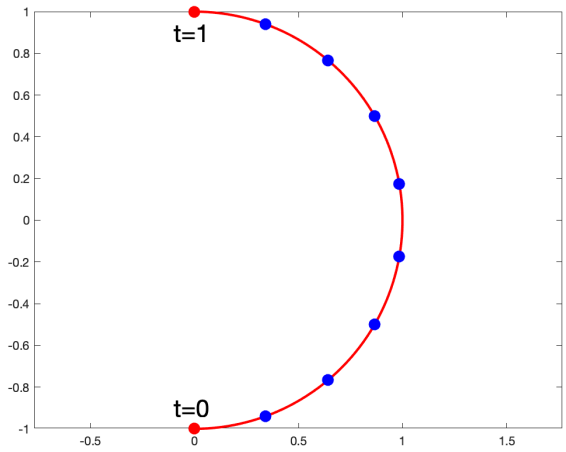


Figure 11.7: The data points in $\mathbf{X}$ do not lie in a one-dimensonal subspace, but they do lie on a semicircle.

where $f(\widetilde{\mathbf{X}}) = \mathbf{X}$ if the notation is taken to mean that f is applied to each row of $\widetilde{\mathbf{X}}$ individually.

The improvement comes with a significant catch: the set of all functions is vastly larger than the set of all linear functions. If we want to have a computer handle the business of fitting a function to a set of data, we have to give the computer a way to decide what sorts of functions it should attempt to fit. Consequently, nonlinear dimension reduction methods can be significantly more complicated than linear methods. We will focus on linear methods for the remainder of this book, but it should be remembered that just because the theory for linear algebra is well-understood does not mean that it is the right tool for every problem.

11.3.2 Principal Component Analysis

The key to solving our dimension reduction problem is to recognize that if $\Pi \in \mathbb{R}^{d\times d}$ is a rank-k projection, then necessarily $\widetilde{X} = \mathbf{X}\Pi$ has rank at most k. As a result, the Eckart-Young Theorem (Theorem 10.3.2) tells us how to find the optimal projection using the singular value decomposition.

Theorem 11.3.2 Given a matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ and target rank $k < d$, let $\mathbf{V}_k$ be a matrix whose columns are the top k right vectors of $\mathbf{X}$ (unique if and only if $\sigma_k > \sigma_{k+1}$). Then $\Pi = \mathbf{V}_k\mathbf{V}_k^T$ is a solution to the problems

$$\max_{\text{rank}(\Pi)=k} \|\mathbf{X}\Pi\|_F, \quad \text{or} \quad \min_{\text{rank}(\Pi)=k} \|\mathbf{X}(\mathbf{I}-\Pi)\|_F.$$

In addition, $\mathbf{V}_k$ solves the problem

$$\max_{\mathbf{Q}} \|\mathbf{X}\mathbf{Q}\|_F$$

where the maximization is over matrices $\mathbf{Q} \in \mathbb{R}^{d\times k}$ with orthonormal columns.

If we consider the Frobenius norm to be a good indicator of the amount of "information" in a set of data, then taking the singular vectors of $\mathbf{X}$ corresponding to the largest singular values will give us optimal linear dimension reduction.

However! Taking the Frobenius norm of $\mathbf{X}$ (think of this as taking the square-root-sum-of-

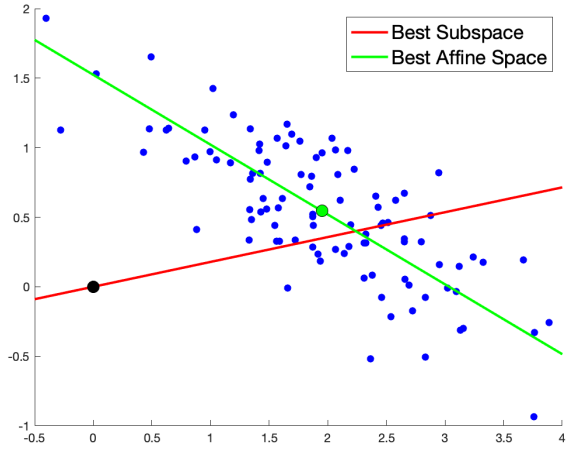


Figure 11.8: The "best" subspace projection (pure SVD) for a set of data versus the best affine space (PCA).

squares of the 2-norms of each row, or data point in $\mathbf{X}$) only makes sense *if we think that the zero vector is a natural point of reference*. In most applications, we are interested in making comparisons *between points in the data set*, so we are far more interested in how far these points are from one another than how far they are from the zero vector.

From a statistical point of view, what we really want to do is maximize the *variation* in the projected data; i.e., the distance of each data point from the center of mass. Ideally, this will make the points as simple to distinguish from one another as possible. From a more mathematical point of view, instead of finding the best *subspace* to project the data onto, we should be looking for the best *affine space*, which does not necessarily contain the zero vector. Fortunately for us, these two approaches coincide. All we have to do is *center the data* first.

Definition 11.3.1 Given a data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ to *center the data* means to subtract from each entry the mean of the entries in its column.

In Matlab, centering the data can be done via the commands `mu = mean(X); X = X-mu`. The vector $\mathbf{m} = [\mu_1, \ldots, \mu_d]$ is a row vector in $\mathbb{R}^d$, and in Matlab the operation `X-mu` is well-defined despite the fact that $\mathbf{X}$ is a matrix and $\mathbf{m}$ is a vector. Since they have the same number of columns, μ_i is subtracted from each element of the ith column for $1 \leq i \leq d$.

The vector of column means $\mathbf{m}$ can be obtained through the matrix-vector product

$$\mathbf{m} = \frac{1}{n}\mathbb{1}^T\mathbf{X},$$

where $\mathbb{1}$ is the all-ones vector. The operation of centering the data can in turn be interpreted as a projection

$$\mathbf{X}_{\text{ctr}} = \mathbf{X} - \mathbb{1}\mathbf{m} = \mathbf{X} - \mathbb{1}\left(\frac{1}{n}\mathbb{1}^T\mathbf{X}\right) = \mathbf{X} - \left(\frac{1}{n}\mathbb{1}\mathbb{1}^T\right)\mathbf{X} = (\mathbf{I} - \Pi_{\mathbb{1}})\mathbf{X}.$$

It would be unnecessarily inefficient to compute either the column means or the centered data in this manner, but the identity can be useful as a theoretical tool. We also need the following fact about orthogonal projections onto affine spaces:

Fact 11.3.3 Given an affine space in $\mathbb{R}^d$ represented as $\mathscr{A} = \mathbf{a} + \mathscr{V}$, where $\mathscr{V}$ is a subspace of $\mathbb{R}^d$ and $\mathbf{a} \in \mathbb{R}^d$, the orthogonal projection onto the affine space $\mathscr{A}$ is given by

$$\Pi_{\mathscr{A}}\mathbf{x} = \Pi_{\mathscr{V}}(\mathbf{x} - \mathbf{a}) + \mathbf{a}.$$

This projection does not depend on the particular choice of $\mathbf{a}$ to represent the affine space: for any $\mathbf{v} \in \mathscr{V}$, the vector $\mathbf{a}' = \mathbf{a} + \mathbf{v}$ would also work. It is also *not* a linear transformation in general, since $\Pi_{\mathscr{A}}\mathbf{0} \neq \mathbf{0}$ if $\mathscr{A}$ does not contain the zero vector.

Fact 11.3.4 In the case of projecting a data matrix $\mathbf{X}$ onto a smaller-dimensional space, the projection would take the form

$$\mathbf{X}\Pi_{\mathscr{A}} = (\mathbf{X} - \mathbb{1}\mathbf{a}^T)\Pi_{\mathscr{V}} + \mathbb{1}\mathbf{a}^T.$$

Again note that $\Pi_{\mathscr{A}}$ is not a linear transformation in general, so the expression $\mathbf{X}\Pi_{\mathscr{A}}$ is a slight abuse of notation.

Back to our original goal: establishing that centering the data unites the two viewpoints. We end up with a theorem that looks something like an affine-space version of Eckart-Young.

Theorem 11.3.5 Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ and target rank $k < d$, the problems

- Find a k-dimensional affine space $\mathscr{A} \subseteq \mathbb{R}^d$ minimizing $\|\mathbf{X} - \mathbf{X}\Pi_{\mathscr{A}}\|_F$,
- Find a k-dimensional subspace $\mathscr{V} \subseteq \mathbb{R}^d$ maximizing $\|\mathbf{X}\Pi_{\mathscr{V}} - \mathbb{1}\mathbf{m}_{\mathscr{V}}\|_F$, where $\mathbf{m}_{\mathscr{V}} = (1/n)\mathbb{1}^T(\mathbf{X}\Pi_{\mathscr{V}})$ contains the column means of $\mathbf{X}\Pi_{\mathscr{V}}$

are both solved by centering the data $\mathbf{X}_{\text{ctr}} = \mathbf{X} - \mathbb{1}\mathbf{m}$ where $\mathbf{m} = (1/n)\mathbb{1}^T\mathbf{X}$, then setting $\mathscr{V}$ to be the subspace spanned by the top singular vectors of $\mathbf{X}_{\text{ctr}}$. The optimal affine space is then $\mathscr{A} = \mathbf{m} + \mathscr{V}$.

Proof. We'll start with the first problem, which has the interpretation "find the affine space that minimizes the distance between the original and projected data". By Fact 11.3.4, we observe that

$$\|\mathbf{X} - \mathbf{X}\Pi_{\mathscr{A}}\|_F = \|\mathbf{X} - \left[(\mathbf{X} - \mathbb{1}\mathbf{a}^T)\Pi_{\mathscr{V}} + \mathbb{1}\mathbf{a}^T\right]\|_F = \|(\mathbf{X} - \mathbb{1}\mathbf{a}^T)(\mathbf{I} - \Pi_{\mathscr{V}})\|_F.$$

Next, we consider the orthogonal decompositions

$$\mathbf{X} = \Pi_{\mathbb{1}}\mathbf{X} + (\mathbf{I} - \Pi_{\mathbb{1}})\mathbf{X}, \qquad \mathbb{1}\mathbf{a}^T = \Pi_{\mathbb{1}}\mathbb{1}\mathbf{a}^T + \mathbf{0},$$

and find that by the Pythagorean Theorem (Corollary 11.2.8),

$$\begin{aligned}
\|(\mathbf{X} - \mathbb{1}\mathbf{a}^T)(\mathbf{I} - \Pi_{\mathscr{V}})\|_F^2 &= \|(\mathbf{I} - \Pi_{\mathbb{1}})\mathbf{X}(\mathbf{I} - \Pi_{\mathscr{V}}) + \Pi_{\mathbb{1}}(\mathbf{X} - \mathbb{1}\mathbf{a}^T)(\mathbf{I} - \Pi_{\mathscr{V}})\|_F^2 \\
&= \|(\mathbf{I} - \Pi_{\mathbb{1}})\mathbf{X}(\mathbf{I} - \Pi_{\mathscr{V}})\|_F^2 + \|\Pi_{\mathbb{1}}(\mathbf{X} - \mathbb{1}\mathbf{a}^T)(\mathbf{I} - \Pi_{\mathscr{V}})\|_F^2 \\
&= \|\mathbf{X}_{\text{ctr}}(\mathbf{I} - \Pi_{\mathscr{V}})\|_F^2 + \|\mathbb{1}(\mathbf{m} - \mathbf{a}^T)(\mathbf{I} - \Pi_{\mathscr{V}})\|_F^2,
\end{aligned}$$

where $\mathbf{m}$ and $\mathbf{X}_{\text{ctr}}$ are defined in the statement of the theorem. By Eckart-Young, the first term is minimized when $\mathscr{V}$ is spanned by the dominant right singular vectors of $\mathbf{X}_{\text{ctr}}$. The second term is equal to zero when $\mathbf{a}^T = \mathbf{m}$.

The second problem has the interpretation "find the projection that maximizes the spread of the projected data about its own center". The term to be maximized can be rewritten as

$$\|\mathbf{X}\Pi_{\mathscr{V}} - \mathbb{1}\mathbf{m}_{\mathscr{V}}\|_F = \|\mathbf{X}\Pi_{\mathscr{V}} - (1/n)\mathbb{1}\mathbb{1}^T\mathbf{X}\Pi_{\mathscr{V}}\|_F = \|(\mathbf{I} - \Pi_{\mathbb{1}})\mathbf{X}\Pi_{\mathscr{V}}\|_F = \|\mathbf{X}_{\text{ctr}}\Pi_{\mathscr{V}}\|_F.$$

Once again, Eckart-Young implies that the optimal $\mathscr{V}$ is spanned by the dominant right singular vectors of $\mathbf{X}_{\text{ctr}}$. ∎

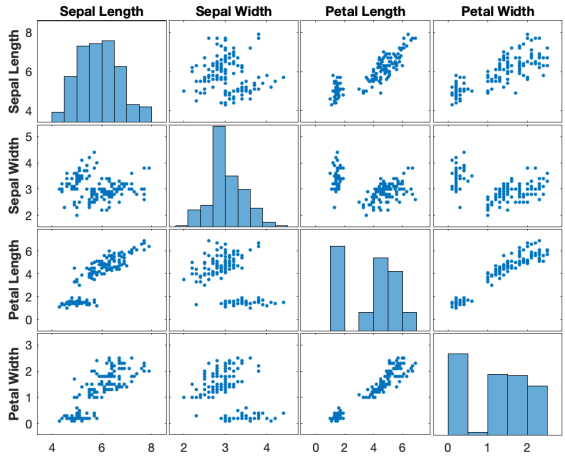


Figure 11.9: Pairwise scatterplots for iris measurements

The result of all of this matrix algebra is a tool that goes by many names and has been rediscovered many times over the last century, but is most commonly known as *principal component analysis*. For now we focus on the SVD-based perspective of this tool, but later on we will revisit it from the point of view of statistics, and in particular random variables.

Definition 11.3.2 — Principal Component Analysis. Given a data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, *principal component analysis* is equivalent to taking the economy-size singular value decomposition of the centered data matrix, $\mathbf{X}_{\text{ctr}} = \mathbf{U}\Sigma\mathbf{V}^T$.

- The *principal components* are the columns $\mathbf{v}_1, \ldots, \mathbf{v}_d$ of $\mathbf{V}$.
- The *coefficients* of a given principal component $\mathbf{v}_i$ are the elements of $\mathbf{v}_i$.
- The *scores* of the jth observation are the jth row of $\mathbf{U}\Sigma$.

For a target rank $k < d$, the dimension-reduced data set is $\mathbf{X}_{\text{ctr}}\mathbf{V}_k$, where $\mathbf{V}_k$ contains the first k columns of $\mathbf{V}$.

11.3.3 Iris Flower Data Set

As an example, we'll take a look at the *Iris flower data set*, introduced by Ronald Fisher in 1936 as an example of a classification problem. The data set consists of 150 Iris flowers with 50 from each of three species, the *versicolor*, *virginica*, and *setosa*. Each flower has four measurements: the sepal length and width, and the petal length and width (all in centimeters). One question we might ask is whether knowledge of these measurements alone can allow us to accurately identify the species of a given flower.

More immediately, we might want to know the most useful way to visualize the data. Each flower can be decribed as a vector in $\mathbb{R}^4$, but it is not so simple to picture points in 4-dimensional space. Figure 11.9 shows scatterplots of each pair of measurements, as well as histograms for each individual measurement. At a glance, it seems like any pair of variables does a decent job separating the data for us. In particular, knowing just the petal length or the petal width appears to give us enough information to tell one species (*setosa*) apart from the other two. But these two measurements are closely related, and taking a linear combination of the two might be even better.

So we center the data and take the SVD, finding that the singular values of the centered matrix are

$$\sigma_1 \approx 25, \quad \sigma_2 \approx 6.0, \quad \sigma_3 \approx 3.4, \quad \sigma_4 \approx 1.9.$$

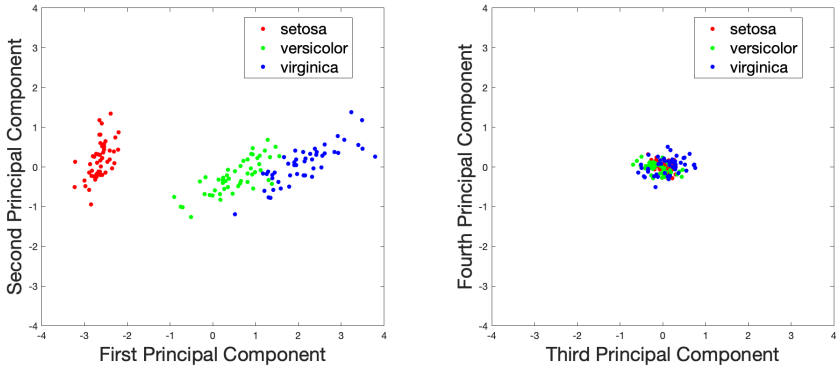


Figure 11.10: The top two principal compnents contain most of the useful information in the data set.

The large gap between σ_1 and the rest suggests that the data set can be fairly well summarized in terms of a single parameter. What parameter is that? Looking at the first principal component, we find that

$$\mathbf{v}_1 \approx \begin{bmatrix} 0.36 \\ -0.08 \\ 0.86 \\ 0.36 \end{bmatrix},$$

which means that the first score (i.e., the x-value of the data points in the first plot of Figure 11.10) is (up to rounding of the coefficients) given by

$$\text{SCORE}_1 = 0.36(\text{sepal length}) - 0.08(\text{sepal width}) + 0.86(\text{petal length}) + 0.36(\text{petal width})$$

The second plot in Figure 11.10 shows that all of the data have small scores in the third and fourth principal components, and that these scores are not very helpful in distinguishing the species. Thus most of the useful information in the data set (as far as classifying the irises is concerned) is contained within the affine space determined by the mean of the data and the first two principal components.

Given the close relationship between petal length and width apparent in Figure 11.9, it is interesting that the petal length is much more aligned with the first principal component. This happens because the petal lengths have a standard deviation of about 1.76, but the petal widths only have a standard deviation of about 0.76. Becuase there is more variation in the former, PCA considers it more important to capture this direction. The lesson here is that scaling your data appropriately matters! If we rescaled the data so that petal length was measured in meters instead (and the rest in centimeters), the new first principal component would be

$$\mathbf{v}_1' \approx \begin{bmatrix} 0.73 \\ -0.12 \\ 0.016 \\ 0.67 \end{bmatrix}.$$

So what's a proper way to scale the data? There isn't any single obviously correct answer. It might make sense to use the same units for each measurement, if possible, as with the Iris data. But for many data sets this won't be possible: for example, a set of hospital data that includes patients' height, weight, and resting heart rate. Without any other knowledge of the data, a reasonable default

Program 11.1: Dimension reduction on Iris data

```
load fisheriris % 'meas' is (150 x 4) data matrix
plotmatrix % show pairs of variables
mu = mean(meas);
meas = meas - mu; % center the data
[U,S,V] = svd(meas,'econ') % could also have used pca(meas)
X = meas*V; % PCA scores, also equal to U*S
figure
subplot(1,2,1)
gscatter(X(:,1),X(:,2),species)
xlabel('First Principal Component','fontsize',20)
ylabel('Second Principal Component','fontsize',20)
xlim([-4,4])
ylim([-4,4])
subplot(1,2,2)
gscatter(X(:,3),X(:,4),species)
xlabel('Third Principal Component','fontsize',20)
ylabel('Fourth Principal Component','fontsize',20)
xlim([-4,4])
ylim([-4,4])
```

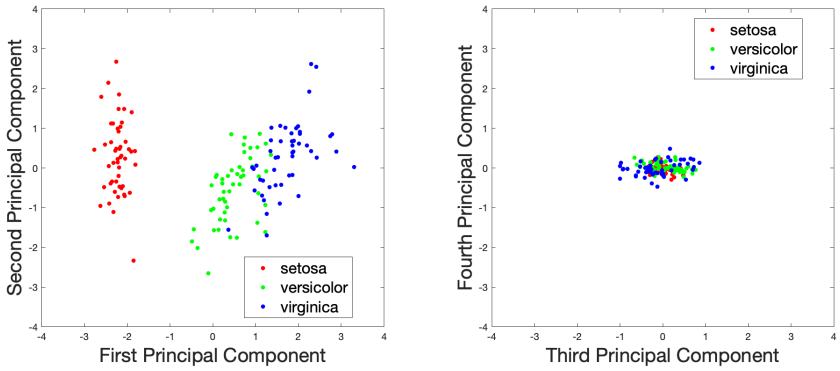


Figure 11.11: PCA after the columns have been rescaled to have unit variance.

method might be to scale the data so that each column has variance equal to 1. Figure 11.11 shows the results of doing this for the Iris data set, and the new first principal component is

$$\mathbf{v}_1'' \approx \begin{bmatrix} 0.52 \\ -0.27 \\ 0.58 \\ 0.56 \end{bmatrix}.$$

In this specific case, rescaling the data seems to make the *versicolor* and *virginica* species more difficult to distinguish from one another by the first two principal components alone. But in general the choice of scaling for PCA is somewhat arbitrary.

11.3.4 MNIST Data Set

As another example, we'll take a look at the *MNIST database*, a collection of handwritten digits stored as 28×28 images (each represented by a vector in $\mathbb{R}^{784}$). The set is commonly used as a benchmark for classification algorithms, where the training set contains sixty thousand images

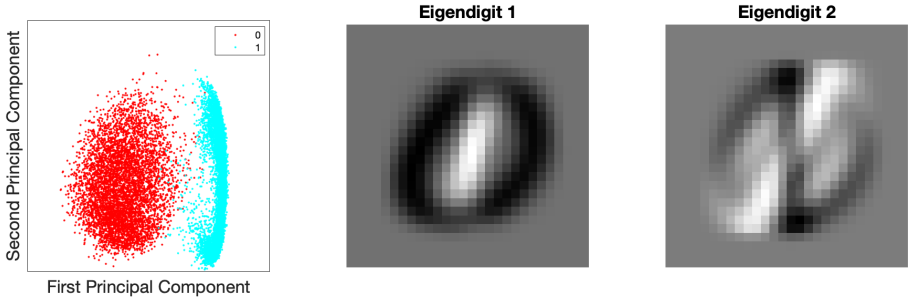


Figure 11.12: Left: Scatterplot of reduced data. Right: Top principal component is very useful for distinguishing the digits 0 and 1; the second component is not so useful.

and the test set contains another ten thousand. In any case, PCA only selects the directions that maximize the variance of the reduced data, and doesn't explicitly account for the classes to which the data points belong. So although we are not *guaranteed* that the top principal components will be useful for distinguishing the different classes of data (in this case, the "class" is the identity of the digit, and the "data" is the image vector), we can at least *hope* that PCA will prove useful for this purpose.

Figure 11.12 shows a successful example: we first take the training data and retain only the digits whose labels are 0 or 1, for a total of 12,665 samples. Then we run PCA and retain the top two principal components. The figure shows that the two digits can be almost perfectly distinguished from one another on the basis of the first principal component alone, and that the second is not quite as useful (it appears to measure something like how much the image is slanted to the right). The directions for those principal components (i.e., right singular vectors) are shifted and scaled to fit the interval $[0, 1]$, then shown as grayscale images (sometimes called *eigendigits*). In the first of these image, lighter pixels represent moving in the positive direction of the first principal component, darker pixels represent moving in the negative direction, and gray pixels are neutral. If we find the scores for a given image, the implication is that images with lots of mass in the dead center have a higher first principal component score, and so is more likely to be a 1.

11.4 Matlab Commands

PCA

- `[COEFF,SCORE] = pca(X)` returns the principal component coefficients and scores for the data matrix `X`. Equivalent in exact arithmetic to `[U,S,V] = svd(X-mean(X),'econ'); COEFF = V; SCORE = U*S`, at least up to a factor of ± 1 in each column.

Plotting

Lines:

- `close`: Close the current figure. `close all` closes all open figures.
- `clf`: Clears the currrent figure.
- `plot(xdata,ydata)`: Creates a line plot from two vectors. The plot does not need to be the graph of a function–instead, it plots all of the (x,y) points and connects adjacent points with a straight line.
- `semilogx(xdata,ydata)`: Creates a line plot where the x-axis is on a logarithmic scale. See also `semilogy` and `loglog`.
- `fplot(fun,lims)`: Plots a graph of the function handle `fun` with in the specified limits, with a default of [-5 5].

- `hold on`: Allows you to plot multiple lines on a single plot. `hold off` turns this setting off.
- `yline(val)`: Creates a constant line at the value `val`. See also `xline`.
- `Color`: Set the color of the line. Colors with short names are 'r', 'g', 'b', 'c' (cyan), 'm' (magenta), 'y', 'k' (black), and 'w'.
- `LineStyle`: Set the style of the line. Options are '-', '–', ':', '-.', and 'none'.
- `LineWidth`: The width of the line. Default is 0.5, where 1pt is 1/72 of an inch.
- `Marker`: Put markers down at the plotted points. Some common options are 'o', '+', and '*'. More detailed directions can be given with `MarkerFaceColor`, `MarkerSize`, `MarkerEdgeColor`, and `MarkerIndices` (if you want only some of the plotted points to have markers).

Labels:

- `title(str)`: Set the title. See also `xlabel` and `ylabel`.
- `FontSize`: The font size.
- `Interpreter`: Set to 'latex' in order to display math equations in the title.

Legend:

- `legend(str1,str2,...)`: Sets a legend with the input strings being the labels for each plotted line (by default, in the order they were plotted). It is possible to set a legend for only a subset of the plotted lines, but doing so is a little more difficult.
- `Location`: Places the legend on the figure. Default is 'northeast'.
- `Orientation`: Choose between a horizontal or vertical legend.
- `FontSize`: The font size.
- `Interpreter`: Set to 'latex' in order to display math equations in the legend.

Axes:

- `xlim([a,b])`: Sets the limits of the x-axis to the interval $[a,b]$. See also `ylim`.
- `axis equal`: Rescales the plot so that equal tick mark increments in the x and y directions are equal in size.
- `axis tight`: Sets the axis limits to the range of the data.
- `grid on/off/minor`: Include major, minor, or no grid lines on the plot.
- `xticks(v)`: Set the x-axis tick marks to the values in the vector `v`. See also `yticks`.

Scatter plots:

- `scatter(xdata,ydata)`: Makes a scatter plot from the data.
- `gscatter(xdata,ydata,G)`: Makes a scatter plot where the data are automatically color-coded by the entries of G. Points with the same value of G are shown with the same color and marker.
- `plotmatrix(X,Y)`: Makes scatter plots of each column of `X` against each column of `Y`.
- `plotmatrix(X)`: Same as `plotmatrix(X,X)`, but diagonal plots are replaced by histograms.
- `Marker`: The type of marker used.
- `SizeData`: The size of the markers.
- `MarkerFaceColor`: The color of the interior of the markers. See also `MarkerEdgeColor`.
- `LineWidth`: The width of the edge of the markers.

Figures:

- `figure`: Opens a new figure.
- `f = figure`: Opens a new figure and stores its handle as the variable `f`. Commands `p = plot(...)`, `p = subplot(...)`, and `lgd = legend(...)` behave similarly.
- `f = gcf`: **G**ets the **C**urrent **F**igure if one is open, and stores its handle as the variable `f`.
- `ax = gca`: **G**ets the **C**urrent **A**xis if one is open, and stores its handle as the variable `ax`.
- `subplot(m,n,p)`: Breaks up the figure into an $m \times n$ grid and puts a subplot in the p-th location (counting across row by row). If `p` is a vector, puts down a subplot that covers all of the indices specified by `p`.

- `print(filename,formattype)` will automatically save your figure with the specified name and format type (such as '-dpdf' or '-dpng' for PDF or PNG).

12. Linear Least Squares

Suppose we're going on a road trip, and partway through we start recording the distance we've traveled:

Time (h)	Distance (mi)
2	123
3	188
4	250
5	310

About how far had we gone 4.5 hours into the trip? If we only want to drive 8 hours per day total, about how far will we make it by the end of the day? Assuming that we're on a long stretch of road with a constant speed limit, it would be fairly reasonable to model our distance y as a linear function of the elapsed time x, subject to some noise. In other words, we want to fit the model

$$y \approx \beta_0 + \beta_1 x$$

to the available data $\{(x_i, y_i)\}_{i=1}^{4}$.

What values of β_0 and β_1 give the "best" approximation, and how do we find them? There are lots of ways we could try to fit a linear model. We could find the line that connects the first and last data points. We could find a line that passes exactly through as many data points as possible. But we're going to take an approach that relies on vector norms instead: if we put the data in matrix-vector form as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix},$$

then we ought to have $\mathbf{y} \approx \mathbf{X}\beta$. This means that the *residual vector*

$$\mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \mathbf{y} - \mathbf{X}\beta$$

Time (h)	Distance (mi)	Predicted Distance (mi)	Residual (mi)
2	123	124.3	-1.3
3	188	186.6	1.4
4	250	248.9	1.1
5	310	311.2	-1.2

Figure 12.1: Actual and predicted values after finding the best fit line for the driving data.

should be as small as possible with respect to some norm. Which norm? Minimizing $\|\mathbf{r}\|_\infty$ will minimize the *error of our least accurate prediction*. Minimzing $\|\mathbf{r}\|_1$ will minimize the *sum of the absolute errors over all of our prections*. But by far the most common choice is to minimize the *sum of the squares of the residuals*, or $\|\mathbf{r}\|_2$. Why? Because the theory is well-understood and the math works out nicely, that's why! It ultimately comes down to the fact that the 2-norm is derived from an inner product (see Section 3.4.1), and so is accompanied by a large number of useful properties such as the Pythagorean Theorem. The choice also mirrors our decision to use the standard deviation rather than the mean absolute deviation as a measure of spread back in Section 5.2.

Generalizing to linear models with more input variables $y \approx \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_n x^{(n)}$, we frame the problem generally as

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2.$$

At least, this is the notation more commonly used in statistics settings. This being a math class, we'll opt to use $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$ in place of $\mathbf{X}$, β, and $\mathbf{y}$.

Definition 12.0.1 For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and vector $\mathbf{b} \in \mathbb{R}^m$, the *linear least squares* problem is

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2. \tag{LS}$$

Definition 12.0.2 In the statistical setting, the least square problem is often called *linear regression*. When fitting a line based on a single input variable, the solution is called the *best fit line* for the data.

Typically, but not always, the matrix $\mathbf{A}$ will have more rows than columns. Generically, we would therefore expect that $\mathbf{b} \notin \mathscr{R}(\mathbf{A})$, or that there is no $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}$. For example, if we have more than two data points, it is unlikely that there exists a single line that passes through all of them. Intuitively, this is because we only have two free parameters that we can use to define a line (e.g., slope and y-intercept), and more than two constraints that must be satisfied.

Definition 12.0.3 A system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is called *overdetermined* if $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m > n$.

■ **Example 12.1** For the road trip problem, solving the least-squares problem gives the linear approximation

$$y \approx -0.3 + 62.3x.$$

Predicted values and residuals $y_i - \hat{y}_i$ are given in Figure 12.1, where $\hat{y} = \beta_0 + \beta_1 x$ denotes the vector of predicted outputs. ■

In Matlab, solving the least-squares problem is as simple as typing `A\b` (well, barring complications). But what does Matlab do to solve the problem? That's the topic for this chapter!

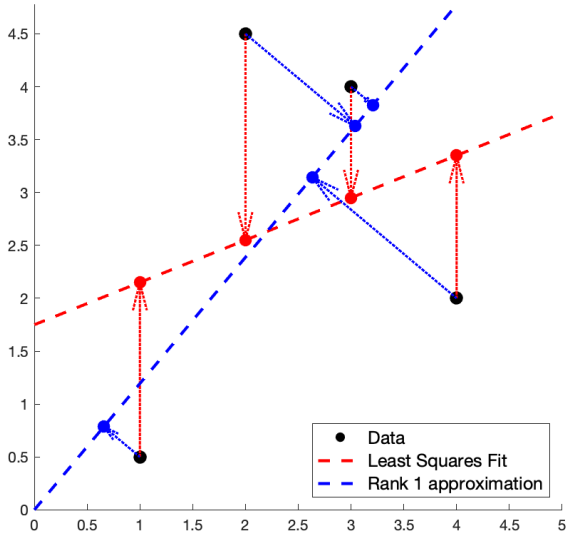


Figure 12.2: PCA versus least-squares on a small data set.

12.0.1 Least Squares versus PCA

For both the least-squares problem and PCA, we attempt to simplify a set of data by making some sort of linear approximation to it. However, these two problems are used in fairly different settings. For the least squares problem, there is one special variable y that we would like to predict as a *function* of the other variables $x^{(1)},\ldots,x^{(n)}$. By solving the least-squares problem we fit a hyperplane to the data that minimizes the sum-of-squares of the *vertical* distances from the data points y_i to their predicted values $\hat{y}_i$, for $1 \leq i \leq m$. PCA, by contrast, does not distinguish any one particular variable apart from the others. It finds an affine space that approximates the data, minimizing the *orthogonal* distance from the data to the space. Depending on the situation, the approximations returned by the two methods might look quite different—see Figure 12.2 as an example. When working with data matrices, though, one interesting feature that the approximations have in common is that they both pass through the center of mass of the data.

12.1 Three Approaches

Here we show how to solve the least-squares problem LS from each of three different perspectives. In the first, we use the language of projections. In the second, we use calculus. In the third, we use the SVD.

12.1.1 Projections

First taking an approach in terms of projections, we observe that by the definition of $\mathscr{R}(\mathbf{A})$,

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2 = \min_{\mathbf{y}\in\mathscr{R}(\mathbf{A})} \|\mathbf{b}-\mathbf{y}\|_2.$$

The insight, supported visually by Figure 12.3, is that the optimal solution to the latter problem is simply $\hat{\mathbf{b}}$, the orthographic projection of $\mathbf{b}$ onto $\mathscr{R}(\mathbf{A})$. Here we are using the orthogonal decomposition (Notation 11.3)

$$\mathbf{b} = \Pi_{\mathbf{A}}\mathbf{b} + (\mathbf{I}-\Pi_{\mathbf{A}})\mathbf{b} = \hat{\mathbf{b}} - \mathbf{b}_{\perp}.$$

To establish this idea formally:

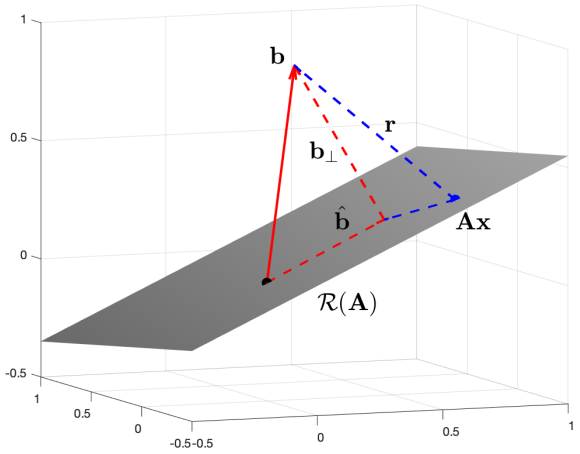


Figure 12.3: The projection of $\mathbf{b}$ is the unique point in $\mathscr{R}(\mathbf{A})$ closest to $\mathbf{b}$.

Theorem 12.1.1 Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and vector $\mathbf{b} \in \mathbb{R}^m$, the vector $\hat{\mathbf{b}} = \Pi_{\mathbf{A}}\mathbf{b}$ is the unique point in $\mathscr{R}(\mathbf{A})$ closest to $\mathbf{b}$ in terms of the 2-norm.

Proof. We start by noting that for any $\mathbf{x} \in \mathbb{R}^n$, by the definitions of $\hat{\mathbf{b}}$ and $\mathbf{b}_\perp$ it holds that

$$\mathbf{b}_\perp^T(\hat{\mathbf{b}} - \mathbf{A}\mathbf{x}) = \mathbf{b}^T(\mathbf{I} - \Pi_{\mathbf{A}})(\Pi_{\mathbf{A}}\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{0},$$

since $(\mathbf{I} - \Pi_{\mathbf{A}})\Pi_{\mathbf{A}} = \mathbf{0}_{m\times m}$ and $(\mathbf{I} - \Pi_{\mathbf{A}})\mathbf{A} = \mathbf{0}_{m\times n}$. It follows by the Pythagorean Theorem that

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{b}_\perp + \hat{\mathbf{b}} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{b}_\perp\|_2^2 + \|\hat{\mathbf{b}} - \mathbf{A}\mathbf{x}\|_2^2.$$

The final expression is minimized when its second term is zero, which occurs if and only if $\mathbf{A}\mathbf{x} = \hat{\mathbf{b}}$. ■

This tells us that $\mathbf{x}$ is a solution if and only if $\mathbf{A}\mathbf{x} = \hat{\mathbf{b}}$. In order to simplify this condition, we make an additional observation.

Theorem 12.1.2 The unique point $\mathbf{y} \in \mathscr{R}(\mathbf{A})$ satisfying $\mathbf{A}^T(\mathbf{b} - \mathbf{y}) = \mathbf{0}$ is $\hat{\mathbf{b}}$.

Proof. First, we show that $\hat{\mathbf{b}}$ satisfies the desired property. By its definition, we find that

$$\mathbf{A}^T(\mathbf{b} - \hat{\mathbf{b}}) = \mathbf{A}^T\mathbf{b}_\perp = \mathbf{A}^T(\mathbf{I} - \Pi_{\mathbf{A}})\mathbf{b} = \mathbf{0}.$$

To show that $\hat{\mathbf{b}}$ is unique, we note that since $\hat{\mathbf{b}} \in \mathscr{R}(\mathbf{A})$ there exists a point $\mathbf{x}_*$ (not necessarily unique) such that $\mathbf{A}\mathbf{x}_* = \hat{\mathbf{b}}$. Then if $\mathbf{x}$ satisfies $\mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{0}$, it follows that

$$\mathbf{0} = \mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{A}^T(\mathbf{b}_\perp + \hat{\mathbf{b}} - \mathbf{A}\mathbf{x}) = \mathbf{A}^T\mathbf{b}_\perp + \mathbf{A}^T(\mathbf{A}\mathbf{x}_* - \mathbf{A}\mathbf{x}) = \mathbf{A}^T\mathbf{A}(\mathbf{x}_* - \mathbf{x}).$$

By Theorem 8.3.5, $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}$ have the same null space, so it follows that

$$\mathbf{0} = \mathbf{A}(\mathbf{x}_* - \mathbf{x}) = \hat{\mathbf{b}} - \mathbf{A}\mathbf{x}.$$

Thus $\mathbf{A}\mathbf{x} = \hat{\mathbf{b}}$, and so uniqueness is proved. ■

To summarize: Theorem 12.1.1 says that $\mathbf{x}$ solve LS if and only if $\mathbf{A}\mathbf{x} = \hat{\mathbf{b}}$, and Theorem 12.1.2 says that $\mathbf{A}\mathbf{x} = \hat{\mathbf{b}}$ if and only if $\mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{0}$. Rewriting this in terms of the residual vector $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$, we get the necessary and sufficient condition

$$\mathbf{A}^T\mathbf{r} = \mathbf{0}.$$

In other words, the residual must be perpendicular (i.e., orthogonal, or *normal*) to the column space of $\mathbf{A}$. By rearranging the expression slightly, we get a condition known as the *normal equations*.

Theorem 12.1.3 — Normal Equations. Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$, the vector $\mathbf{x} \in \mathbb{R}^n$ is a solution to $\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$ if and only if $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$.

This least squares problem always has at least one solution since $\hat{\mathbf{b}} \in \mathscr{R}(\mathbf{A})$, but from Theorem 12.1.1 we can deduce more than that.

Theorem 12.1.4 Let $\mathbf{x}_*$ be any solution to $\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$. Then the set of all solutions is $\mathbf{x}_* + \mathscr{N}(\mathbf{A})$.

Proof. A vector $\mathbf{x}$ solves LS if and only if $\mathbf{A}\mathbf{x} = \hat{\mathbf{b}} = \mathbf{A}\mathbf{x}_*$, which is equivalent to $\mathbf{A}(\mathbf{x} - \mathbf{x}_*) = \mathbf{0}$, which in turn is equivalent to $\mathbf{x} - \mathbf{x}_* \in \mathscr{N}(\mathbf{A})$. ■

So the solution is unique if and only if $\mathscr{N}(\mathbf{A}) = \{\mathbf{0}\}$, or equivalently if $\mathbf{A}$ has full column rank. Relatedly, the fact that $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}$ have the same null space means that $\mathbf{A}^T\mathbf{A}$ is invertible if and only if $\mathbf{A}$ has full column rank. Applying this idea to the normal equations, we end up with the following conclusion.

Corollary 12.1.5 If $\mathbf{A}$ has full column rank, then the unique solution to LS is $\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$.

From the proof of Theorem 12.1.1, we also get the following fact:

Fact 12.1.6 Let $\mathbf{x}_*$ be any solution to LS. Then $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{b}_\perp\|_2^2 + \|\mathbf{A}(\mathbf{x}_* - \mathbf{x})\|_2^2$.

12.1.2 Calculus

A second approach is to consider the squared residual norm $f(\mathbf{x}) = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2$ as a function of the inputs $\mathbf{x}$, and to set the gradient to zero. To start, we note that for $\mathbf{A} \in \mathbb{R}^{m\times n}$,

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2 = \sum_{i=1}^m (b_i - (\mathbf{A}\mathbf{x})_i)^2 = \sum_{i=1}^m \left(b_i - \sum_{j=1}^n a_{ij}x_j\right)^2.$$

Taking the partial derivative with respect to the input x_k, we find that

$$\frac{\partial f}{\partial x_k}(\mathbf{x}) = \sum_{i=1}^m 2\left(b_i - \sum_{j=1}^n a_{ij}x_j\right)(-a_{ik}) = -2\sum_{i=1}^m a_{ik}(\mathbf{b} - \mathbf{A}\mathbf{x})_i = -2\mathbf{a}_k^T(\mathbf{b} - \mathbf{A}\mathbf{x}).$$

Doing the same for all indices $1 \le k \le n$ shows that the gradient is

$$\nabla f(\mathbf{x}) = -2\mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}).$$

The gradient is equal to zero if and only if the normal equations are satisfied.

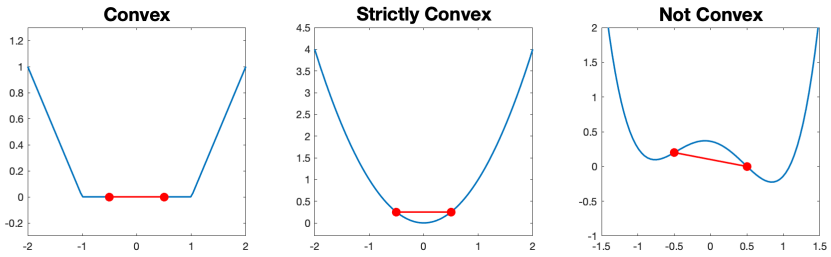


Figure 12.4: Any local minimum of a convex function is also a global minimum.

Theorem 12.1.7 Let $f(\mathbf{x}) = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2$. Then $\mathbf{x}$ is a critical point of f if and only if $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$.

But that's not enough to establish that $f(\mathbf{x})$ is *minimized*; we also need to examine the second derivative. Looking at the matrix of mixed partial derivatives, we find that

$$\frac{\partial^2 f}{\partial x_k \partial x_\ell}(\mathbf{x}) = \frac{\partial f}{\partial x_\ell}\left(\frac{\partial f}{\partial x_k}\right)(\mathbf{x}) = \frac{\partial f}{\partial x_\ell}\sum_{i=1}^{m} 2\left(b_i - \sum_{j=1}^{n} a_{ij}x_j\right)(-a_{ik}) = 2\sum_{i=1}^{m} a_{ik}a_{i\ell} = 2(\mathbf{A}^T\mathbf{A})_{k\ell}.$$

Thus $\nabla^2 f(\mathbf{x}) = 2\mathbf{A}^T\mathbf{A}$. The fact that $\mathbf{z}^T\mathbf{A}^T\mathbf{A}\mathbf{z} = \|\mathbf{A}\mathbf{z}\|_2^2 \geq 0$ for all $\mathbf{z}$ implies that f is a *convex* function. We won't go too much into the theory of convex functions, but suffice it to say that minimizing a convex function is much simpler than minimizing a non-convex one.

Definition 12.1.1 A function $f(\mathbf{x})$ is *convex* if for all $t \in [0,1]$ and all $\mathbf{x}$ and $\mathbf{y}$,

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}).$$

If the inequality is strict everywhere, then f is *strictly convex*.

Visually, this means that if you take any two points on the graph of f, the line segment connecting them falls above (or strictly above) the graph of the function. Figure 12.4 provides an example.

One very nice property of convex functions is that every critical point is a global minimizer of the function. Furthermore, if the function is strictly convex then there is at most one critical point (although there might not be any, such as with e^x).

Fact 12.1.8 If f is convex and $\nabla f(\mathbf{x}) = \mathbf{0}$, then $\mathbf{x}$ is a global minimizer of f. If f is strictly convex, then $\mathbf{x}$ is the *unique* global minimizer of f.

Convex functions also have the property that any local minimum is necessarily a global minimum. Figure 12.4 shows that non-convex functions might not have this property.

Fact 12.1.9 If $\mathbf{A}$ has full column rank, then $\mathbf{z}^T\mathbf{A}^T\mathbf{A}\mathbf{z} > 0$ for all nonzero $\mathbf{z}$, which implies that $f(\mathbf{x}) = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2$ is strictly convex and that the solution to LS is unique.

Thus we reach the same conclusion as before: $\mathbf{x}$ is a solution to LS if and only if it satisfies $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$, and the solution is unique if and only if $\mathbf{A}$ has full column rank.

12.1.3 SVD

On the other hand, we've already done so much work building the theory for the singular value decomposition that it would be a shame not to put it to good use. So let $\mathbf{A}$ have rank r and the

compact SVD

$$\mathbf{A} = \underbrace{\begin{bmatrix}\mathbf{U}_r & \mathbf{U}_\perp\end{bmatrix}}_{\mathbf{U}} \begin{bmatrix}\Sigma_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} \underbrace{\begin{bmatrix}\mathbf{V}_r & \mathbf{V}_\perp\end{bmatrix}^T}_{\mathbf{V}},$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are orthogonal. First, since orthogonal transformations preserve the 2-norm, we find that

$$\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2 = \|\mathbf{U}^T(\mathbf{U}_r\Sigma_r\mathbf{V}_r^T)\mathbf{x}-\mathbf{U}^T\mathbf{b}\|_2 = \left\|\begin{bmatrix}\Sigma_r\mathbf{V}_r^T\mathbf{x}-\mathbf{U}_r^T\mathbf{b} \\ -\mathbf{U}_\perp^T\mathbf{b}\end{bmatrix}\right\|_2. \tag{12.1}$$

The bottom term $-\mathbf{U}_\perp^T\mathbf{b}$ cannot be eliminated, but the top term can be set to zero. We decompose $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{V}\mathbf{y} = \mathbf{V}_r\mathbf{y}_1 + \mathbf{V}_\perp\mathbf{y}_2,$$

and continuing from (12.1) find that

$$\Sigma_r\mathbf{V}_r^T\mathbf{x}-\mathbf{U}_r^T\mathbf{b} = \Sigma_r\mathbf{V}_r^T(\mathbf{V}_r\mathbf{y}_1+\mathbf{V}_\perp\mathbf{y}_2)-\mathbf{U}_r^T\mathbf{b} = \Sigma_r\mathbf{y}_1-\mathbf{U}_r^T\mathbf{b} \qquad -\mathbf{U}_\perp^T\mathbf{b}.$$

Thus $\mathbf{y}_1 = \Sigma_r^{-1}\mathbf{U}_r^T\mathbf{b}$, and so in terms of the SVD:

Theorem 12.1.10 The set of all solutions to LS is given by

$$\mathbf{x} = \mathbf{V}_r\Sigma_r^{-1}\mathbf{U}_r^T\mathbf{b} + \mathbf{V}_\perp\mathbf{y}_2, \qquad \mathbf{y}_2 \in \mathbb{R}^{n-r}.$$

The free variable $\mathbf{y}_2$ selects an element in the null space of $\mathbf{A}$, since the columns of $\mathbf{V}_\perp$ form an orthonormal basis for $\mathcal{N}(\mathbf{A})$. Once again, if $\mathbf{A}$ has full column rank (i.e., $r = n$) then the solution is unique.

12.2 Pseudoinverse

Now that last expression using the SVD seems interesting enough that it deserves its own name.

Definition 12.2.1 If a matrix $\mathbf{A}$ has the compact SVD $\mathbf{U}_r\Sigma_r\mathbf{V}_r^T$, the *pseudoinverse* or *Moore-Penrose inverse* of $\mathbf{A}$ is given by

$$\mathbf{A}^+ = \mathbf{V}_r\Sigma_r^{-1}\mathbf{U}_r^T.$$

This generalizes the notion of a matrix inverse. $\mathbf{A}$ not be invertible (and it cannot be if it is rectangular), but in a sense the pseudoinverse comes as close to it as we can get.

Fact 12.2.1 The pseudoinverse satisfies the following properties:

- If $\mathbf{A}$ is invertible, then $\mathbf{A}^+ = \mathbf{A}^{-1}$.
- $\mathbf{A}^+\mathbf{A} = \mathbf{V}_r\mathbf{V}_r^T = \Pi_{\mathbf{A}^T}$.
- If $\mathbf{A}$ has full column rank, then $\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ and $\mathbf{A}^+\mathbf{A} = \mathbf{I}$.
- $\mathbf{A}\mathbf{A}^+ = \mathbf{U}_r\mathbf{U}_r^T = \Pi_{\mathbf{A}}$.
- If $\mathbf{A}$ has full row rank, then $\mathbf{A}^+ = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$ and $\mathbf{A}\mathbf{A}^+ = \mathbf{I}$.
- If $\mathbf{A}$ has full column rank, then $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ is the unique solution to LS.

It is worth remarking that if $\mathbf{A}$ is rectangular, $\mathbf{A}\mathbf{A}^+$ and $\mathbf{A}^+\mathbf{A}$ cannot both be equal to the identity matrices of their respective sizes. If $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m > n$, for example, then $\operatorname{rank}(\mathbf{A}\mathbf{A}^+) \le n < m$, so it cannot be the case that $\mathbf{A}\mathbf{A}^+ = \mathbf{I}_m$. As mentioned above, this product is at least the projection onto the row space of $\mathbf{A}$, and we get a consolation prize as well.

Fact 12.2.2 For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the pseudoinverse $\mathbf{A}^+$ solves both

$$\min_{\mathbf{X}} \|\mathbf{AX} - \mathbf{I}_m\|_F \quad \text{and} \quad \min_{\mathbf{Y}} \|\mathbf{YA} - \mathbf{I}_n\|_F.$$

So in a sense, the pseudoinverse comes as close to being an inverse of $\mathbf{A}$ as we can reasonably ask.

12.2.1 Rank-Deficient Problems

What if $\mathbf{A}$ does not have full column rank? As mentioned a few times, the solution to the least squares problem will not be unique. However, the pseudoinverse is still interesting in that among all possible solutions, it gives us the one with least 2-norm.

Theorem 12.2.3 Given $\mathbf{A}$ and $\mathbf{b}$, the pseudoinverse solution $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ has the smallest 2-norm of any solution to LS.

Proof. For any solution $\mathbf{x}$ to the least squares problem, Theorem 12.1.10 implies that

$$\mathbf{x} = \mathbf{A}^+\mathbf{b} + \mathbf{z}$$

for some $\mathbf{z} \in \mathcal{N}\mathbf{A}$. Then by the Pythagorean Theorem,

$$\|\mathbf{x}\|_2^2 = \|\mathbf{A}^+\mathbf{b}\|_2^2 + \|\mathbf{z}\|_2^2,$$

and the norm is minimized only when $\mathbf{z} = \mathbf{0}$. ■

Matlab's backslash does not automatically give this solution, but the function `lsqminnorm` does.

■ **Example 12.2** Consider the two least squares problems arising from the inputs

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -0.3 & 0.2 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.25 \\ 0.3 & -0.15 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.3 \end{bmatrix}.$$

Figure 12.5 show contour and surface plots of $\|\mathbf{b} - \mathbf{Ax}\|_2^2$, where the black dots and dashed black lines represent solutions. Since $\mathbf{A}_1$ has full column rank the solution is unique. Since $\mathbf{A}_2$ is rank deficient, the set of solutions is a line. The point on the line closest to zero is given by `lsqminnorm(A,b)`. ■

12.3 Practical Computation

For modestly sized problems where $\mathbf{A}$ does not have any notable structure, the normal equations are the fastest way to solve the least-squares problem. Computing $\mathbf{A}^T\mathbf{A}$ costs about mn^2 flops for $\mathbf{A} \in \mathbb{R}^{m\times n}$, where we can save a factor of 2 over regular matrix-matrix multiplication (Section 8.1.3). Computing $\mathbf{A}^T\mathbf{b}$ is relatively cheap (only $\mathcal{O}(mn)$ flops), and from there solving the system $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$ by Gaussian elimination (technically, useing the Cholesky factorization of Definition 8.1.3) costs about $\frac{1}{3}n^3$ flops. This is significantly cheaper than the SVD.

Using the normal equations comes at a cost, however. It requires us to form $\mathbf{A}^T\mathbf{A}$ explicitly, which has singular values $\sigma_1(\mathbf{A})^2, \ldots, \sigma_n(\mathbf{A})^2$. The condition number is therefore

$$\kappa(\mathbf{A}^T\mathbf{A}) = \frac{\sigma_1^2}{\sigma_n^2} = \left(\frac{\sigma_1}{\sigma_n}\right)^2 = \kappa(\mathbf{A})^2,$$

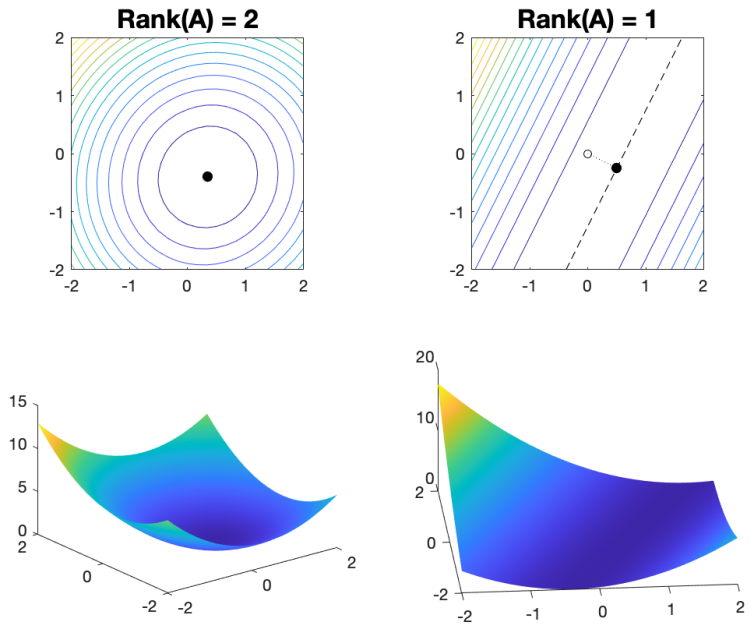


Figure 12.5: Left: full column rank implies unique solution. Right: pseudoinverse gives the least-norm solution.

or the square of the condition number of $\mathbf{A}$. If $\mathbf{A}$ is well-conditioned then this may not pose much of a problem, but if $\kappa(\mathbf{A})$ is large (say, 10^6 or so) then using the normal equations may produce an unacceptably inaccurate answer.

In general, do not solve the least-squares problem by computing the SVD or the pseudoinverse; those methods are slow. In practice, Matlab's default is to use the QR factorization (Section 9.1), which is faster than the SVD and slower but more accurate than the normal equations.

If $\mathbf{A}$ is very large and perhaps very sparse, the $\mathcal{O}(mn^2)$ cost of computing a QR factorization (or SVD, or solving the normal equations) may be prohibitively expensive. In this situation, there exist *iterative methods* which produce a sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots$ that converges to the correct solution. Many of these methods require only a pair of matrix-vector products $\mathbf{Av}$, $\mathbf{A}^T\mathbf{u}$ per iteration, so if the user only wants a few digits of accuracy then such methods may be preferable to the *direct methods* that were covered in this chapter. One of the better-known iterative methods is called LSQR, which is built into Matlab as the function `lsqr`.

12.4 Applications

The most clear-cut application of the least squares problem is linear regression: fitting a line or hyperplane to a set of data where one variable is predicted as a function of the others. Here we discuss a few others.

12.4.1 Arithmetic Mean

One nice feature of the arithmetic mean is that it can be viewed as the solution to a minimization problem.

Fact 12.4.1 For a set of data $\{x_i\}_{i=1}^n$, the arithmetic mean $\bar{x} = \frac{1}{n}\sum_{i=1}^n$ minimizes $\sum_{i=1}^n (x - x_i)^2$.

Proof. By expresing the data as a vector $\mathbf{x}$, the problem may be rewritten in the form

$$\min_x \|\mathbb{1}x - \mathbf{x}\|_2,$$

where $\mathbb{1} \in \mathbb{R}^{n\times 1}$ is the all-ones matrix. Having only a single column, it necessarily has full column rank. We can therefore use the normal equations to find that the unique minimizer x is given by

$$x = (\mathbb{1}^T\mathbb{1})^{-1}(\mathbb{1}^T\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}.$$

■

The median of the data set solves the minimization problem $\min_x \|\mathbb{1}x - \mathbf{x}\|_1$, although the objective is not a differentiable function of x and the solution may not be unique. The minimization problem $\min_x \|\mathbb{1}x - \mathbf{x}\|_\infty$ has the solution $x = (\max_i x_i + \min_i x_i)/2$, which is simple to compute but far more sensitive to outliers in the data.

12.4.2 Polynomial Fitting

We can extend the methods of linear regression to fit higher-degree polynomials to data as well. If we use the model

$$y = \alpha_k x^k + \alpha_{k-1}x^{k-1} + \cdots + \alpha_0,$$

then we can interpret y as a linear function of the inputs $(x, x^2, \ldots, x^k)$ and get the least squares problem

$$\mathbf{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^k \\ 1 & x_2 & \cdots & x_2^k \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^k \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_k \end{bmatrix}.$$

Note that the matrix $\mathbf{A} \in \mathbb{R}^{m\times(k+1)}$ is square if the number of data points m is one larger than the degree k. Such a square matrix is called a *Vandermonde* matrix, and it turns out to be invertible as long as all of the x-values $\{x_i\}_{i=1}^m$ are distinct. This means that two points uniquely determine a line, three points uniquely determine a quadratic, and so on.

■ **Example 12.3** A missile is fired from enemy territory and its position is observed by radar tracking devices as follows:

Position (mi)	0	250	500	750	1000
Height (mi)	0	8	15	19	20

If our goal is to predict where the missile will land, we try to fit a quadratic function through the dataset, getting

$$\mathbf{b} = \begin{bmatrix} 0 \\ 8 \\ 15 \\ 19 \\ 20 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 250 & 250^2 \\ 1 & 500 & 500^2 \\ 1 & 750 & 750^2 \\ 1 & 1000 & 1000^2 \end{bmatrix}.$$

Solving the least squares problem yields the coefficients

$$\alpha_0 \approx -0.2286, \quad \alpha_1 \approx 0.03983, \quad \alpha_2 \approx -1.9429 \cdot 10^{-5},$$

and the larger root of the polynomial (i.e., the predicted landing site) is about 2044.24. Figure 12.6 illustrates the predicted trajectory and landing site for this problem. ■

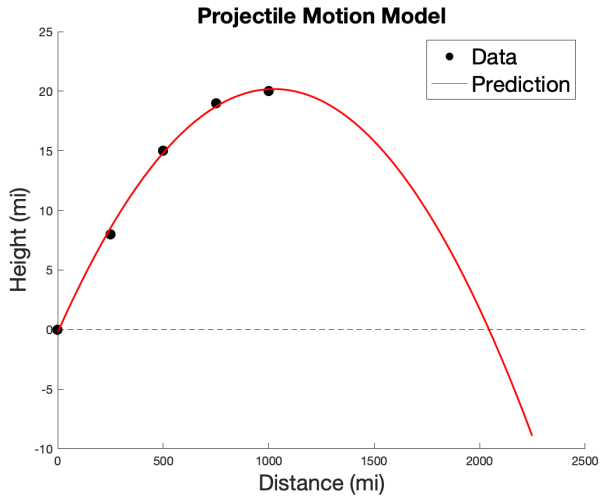


Figure 12.6: Observed data and predicted trajectory of a missile.

In practice, the matrix $\mathbf{A}$ can become very badly conditioned if the degree of the polynomial is too large. Specialized techniques (such as using a different polynomial bases) may be required in this situation.

12.4.3 Exponential Curves and Power Laws

Suppose we have a set of data that we expect to grow at a roughly exponential rate:

$$y \approx \alpha \cdot \exp(\beta x).$$

Since y is not linearly dependent on α and β, we cannot directly use least squares to fit these coefficients. In any case, we might not want to since we should not expect the residual to have approximately the same spread everywhere. If we are modeling population growth, then being off by 100 means a great deal when the population is 200 but not very much when the population is over a million. To resolve these issues, we take the natural logarithm of both sides, getting the transformed model

$$\ln(y) \approx \ln(\alpha) + \beta x.$$

We can now use least squares to approximate $\ln(y)$ as a linear function of x, getting coefficients $\ln(\alpha)$ and β.

■ **Example 12.4** Suppose we want to estimate how quickly the novel coronavirus COVID-19 has been spreading in the US. At least in the early stages of the progression of the virus, it is not unreasonable to model its spread with exponential growth. We look at the data after 100 cases have been confirmed, and after taking logarithms and solving the least squares problem end up with a growth factor of $\beta \approx 0.287$ (Figure 12.7). One way of interpreting this parameter is that the number of cases in the US through the month of March doubled roughly once every $\ln(2)/\beta \approx 2.4$ days. ■

Similarly, if one has the power law

$$y \approx \alpha x^{\beta},$$

then taking the logarithm of both sides gives the transformed model

$$\ln(y) \approx \ln(\alpha) + \beta \ln(x),$$

and from there we can approximate $\ln(y)$ as a linear function of $\ln(x)$.

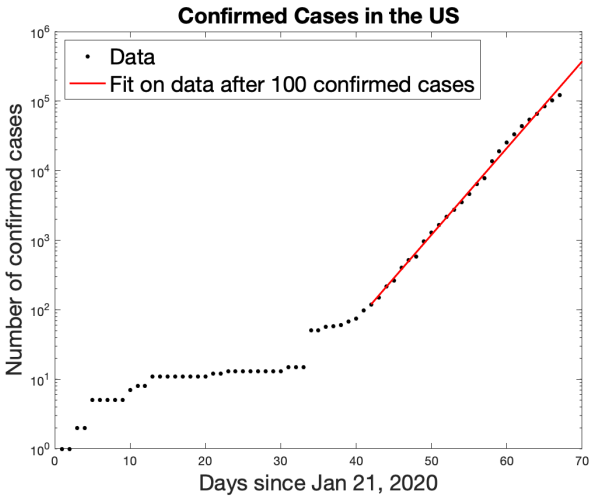


Figure 12.7: Through March 2020, the number of confirmed cases in the US doubled roughly every 2.4 days.

■ **Example 12.5** Body mass index (BMI) uses the formula

$$\text{BMI} = \frac{\text{weight (kg)}}{\text{height}^2\ \text{(m)}}$$

to determine whether a person is overweight or underweight. So what constitutes a healthy BMI? We could try to answer this by fitting the quadratic model

$$w \approx \beta h^2,$$

where w is a person's weight, h the height, and β the coefficient representing BMI. The best fit coefficient for β might represent a "healthy" BMI.

So far, no power law—this is just a polynomial fit. But is 2 really the right power of h to use? The measure for BMI was developed in the 1840's where (due to the lack of calculators) the simplicity of the formula was an important consideration, but in practice the formula will suggest that short people are thinner than they really are and that tall people are fatter than they really are [22]. To get a better fit, we could use the power law model

$$w \approx \beta h^\gamma,$$

and solve for both β and γ. In a 2013 letter, the mathematician Nick Trefethen proposed using the modified formula

$$\text{BMI} = 1.3\frac{\text{weight (kg)}}{\text{height}^{2.5}\ \text{(m)}}.$$

■

It should be noted that the focus of this textbook is on the *mathematics* of least square problems, not on statistics, modeling, or inference. Just because you *can* fit a set of data using linear regression or a power law does not necessarily mean that it is sensible to do so. BMI is a particularly prominent example of a model that can give misleading information when used too carelessly, although there are good theoretical reasons for fitting an exponential model to the COVID-19 case rate data, it is not so simple to predict ahead of time the exponential trend will continue. In this world of fitting statistical models, a little knowledge can be a dangerous thing!

12.5 Matlab Commands

Least Squares

- `A\b`: Solves the least squares problem using QR factorization.
- `lsqminnorm`: Returns the minimum-norm solution to the least squares problem; useful when `A` is rank deficient.
- `lsqr(A,b)`: Iterative methods for solving the least squares problem; useful when `A` is large and sparse. `lsqr(A,b,TOL)` allows the user to specify the desired the accuracy of the solution.
- `pinv(A)`: Pseudoinverse of `A`. `pinv(A,TOL)` treats singular values of `A` below `TOL` as zero; consequently less sensitive to rounding error or noise in the data.
- `polyfit`: In practice, use this for polynomial fitting rather than setting up a least squares problem.

13. Inverse Problems

In many practical situations, we will have to deal with noise or uncertainty in our data. Perhaps some of it is due to human error or measurement error in our tools; perhaps it comes from another source. In any case, we are faced with the problem

$$\mathbf{A}\mathbf{x} = \tilde{\mathbf{b}} = \mathbf{b} + \mathbf{e},$$

where $\mathbf{e}$ represents the error or noise in our measurments for $\mathbf{b}$. The most straightforward answer is to just compute $\tilde{\mathbf{x}} = \mathbf{A}^{-1}\tilde{\mathbf{b}}$, but in many scenarios this may result in an unsatisfactory solution. The question of this chapter is: can we do better?

13.1 Denoising

Let's start with a simpler version of this problem: if an image has been corrupted by noise, to what level of accuracy can we expect to recover the original? If we know absolutely nothing about the structure of the image, then there's not much we can do. But in practice, we can expect some amount of structure: pixels close to one another are likely to have similar color values, and the image may have a decent low-rank approximation. We can use these assumptions of structure to our advantage.

Figure 13.1 shows two common types of noise: *salt-and-pepper noise*, in which pixels may be randomly given values of 0 or 1, and *Gaussian noise*, where the perturbations to each pixel are modeled as indepedent Gaussian random variables. Note that the Gaussian noise makes the problem of *boundary detection* particularly difficult. How can we train a computer to determine where the edges of the inner square lie?

For salt-and-pepper noise or (for low to moderate levels of noise) Gaussian noise, one fairly effective type of filter is a *median filter*, where each pixel is simply replaced by the median of its neighbors. Especially in the case of salt-and-pepper, the idea is that if a defective pixel is surrounded by pixels of a single value, then the defective pixel is probably supposed to have the same value. Figure 13.2 shows the results of using a median filter on both types of noise.

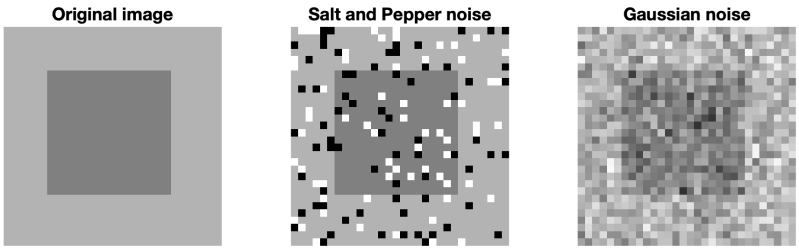


Figure 13.1: Examples of noisy images

13.1.1 Convolutional Filter

The focus of this book is linear algebra, however, so we'll turn to a different type of filter. A *convolution filter* replaces each pixel with a weighted average of its neighbors.

Definition 13.1.1 Given vectors $\mathbf{h} \in \mathbb{R}^{n_h}$ and $\mathbf{x} \in \mathbb{R}^{n_x}$, the (full) convolution $\mathbf{h} * \mathbf{x}$ is a vector in $\mathbb{R}^{n_h+n_x-1}$ whose coefficients are given by

$$(\mathbf{h} * \mathbf{x})_i = \sum_{j=1}^{n_h} h_i x_{j+1-i}, \quad 1 \le i \le n_h + n_x - 1,$$

where indices outside of the range of $\mathbf{h}$ or $\mathbf{x}$ are treated as zero. In the context of image processing, we may call $\mathbf{x}$ the *image* and $\mathbf{h}$ the *kernel*, *convolution matrix*, or *mask*.

It is shown in Section 8.2.3 that multiplication of polynomials is equivalent to convolution of vectors, and that the convolution $\mathbf{b} * \mathbf{a}$ can be represented as a matrix-vector product $\mathbf{b} * \mathbf{a} = \mathbf{B}\mathbf{a}$, where $\mathbf{B}$ is a Toeplitz matrix with the entries of $\mathbf{b}$ along its diagonals.

■ **Example 13.1** Let $\mathbf{h} = \frac{1}{2}[1,1]$ and $\mathbf{x} = [1,2,6,7,10]$. Then

$$\mathbf{h} * \mathbf{x} = \left[\frac{1+0}{2}, \frac{1+2}{2}, \frac{2+6}{2}, \frac{6+7}{2}, \frac{7+10}{2}, \frac{10+0}{2}\right] = [0.5, 1.5, 4, 6.5, 8.5, 5].$$

■

■ **Example 13.2** If $\mathbf{h} = [1]$, then $\mathbf{h} * \mathbf{x} = \mathbf{x}$. ■

In practice, it is often desirable to have the output retain the same dimensions as $\mathbf{x}$. To do this we could simply crop off the leading and trailing elements of $\mathbf{h} * \mathbf{x}$, although we might also want to employ a different method to handle the boundaries of $\mathbf{x}$.

Definition 13.1.2 Given matrices $\mathbf{H} \in \mathbb{R}^{m_h \times n_h}$ and $\mathbf{X} \in \mathbb{R}^{m_x \times n_x}$, the (full) convolution $\mathbf{H} * \mathbf{X}$ is a matrix in $\mathbb{R}^{(m_h+m_x-1)\times(n_h+n_x-1)}$ whose coefficients are given by

$$(\mathbf{H} * \mathbf{X})_{ij} = \sum_{k=1}^{n_h} \sum_{\ell=1}^{m_h} h_{k\ell} x_{k+1-i,\ell+1-j}, \quad 1 \le i \le m_h + m_x - 1, \quad 1 \le j \le n_h + n_x - 1,$$

where indices outside the range of $\mathbf{H}$ or $\mathbf{X}$ are treated as zero. If $m_h = m_x = 1$ or $n_h = n_x = 1$, this definition coincides with Definition 13.1.1.

■ **Example 13.3** Let

$$\mathbf{H} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

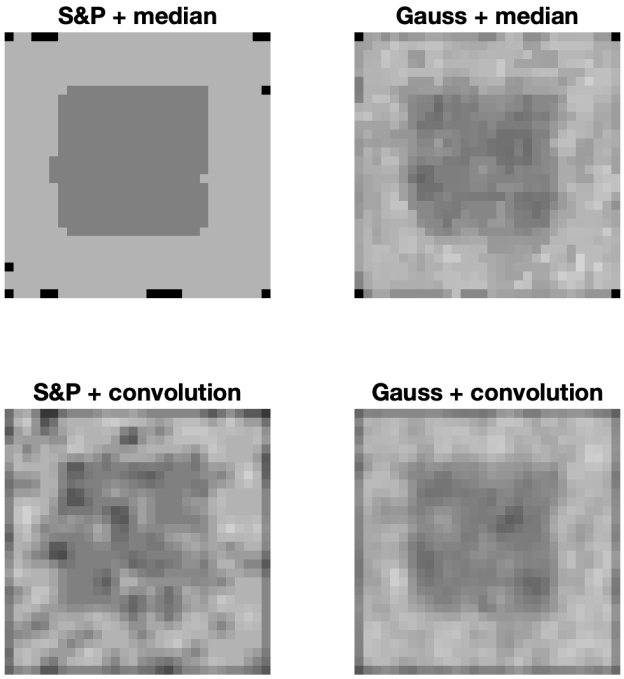


Figure 13.2: Top: median filter. Bottom: Gaussian filter.

Then

$$\mathbf{H} * \mathbf{X} = \begin{bmatrix} 0 & -1 & -4 & -7 & 0 \\ -1 & -1 & 7 & 23 & -7 \\ -2 & 1 & 5 & 19 & -8 \\ -3 & 7 & 13 & 31 & -9 \\ 0 & -3 & -6 & -9 & 0 \end{bmatrix}.$$

If the result were cropped to keep the same shape as $\mathbf{X}$, the natural choice would give the result

$$(\mathbf{H} * \mathbf{X})_{\text{crop}} = \begin{bmatrix} -1 & 7 & 23 \\ 1 & 5 & 19 \\ 7 & 13 & 31 \end{bmatrix}.$$

■

■ **Example 13.4** The second row of Figure 13.2 shows the effect of using the convolutional filter

$$\mathbf{H} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

on noisy image data. ■

Fact 13.1.1 Convolution is commutative: $\mathbf{H} * \mathbf{X} = \mathbf{X} * \mathbf{H}$.

13.1.2 Gaussian Blur

It is relatively straightforward to check from the definition that convolution with a fixed kernel $\mathbf{H}$ is a linear transformation.

Fact 13.1.2 Convolution with a given kernel is a linear transformation: $\mathbf{H} * (\alpha\mathbf{X} + \beta\mathbf{Y}) = \alpha(\mathbf{H} * \mathbf{X}) + \beta(\mathbf{H} * \mathbf{Y})$ in general.

In theory, the convolution $\mathbf{H} * \mathbf{X}$ could therefore be represented as a matrix-vector product

$$\operatorname{vec}(\mathbf{H} * \mathbf{X}) = \operatorname{convMatrix}(\mathbf{H}) \operatorname{vec}(\mathbf{X}),$$

where $\operatorname{convMatrix}(\mathbf{H}) \in \mathbb{R}^{(m_h+m_x-1)(n_h+n_x-1)\times m_x n_x}$ and $\operatorname{vec}(\mathbf{X}) \in \mathbb{R}^{m_x n_x}$, the latter being the matrix $\mathbf{X}$ strung out as a single long vector. In practice, it is probably inefficient to do this.

Fact 13.1.3 To compute $\mathbf{H} * \mathbf{X}$ requires $\mathcal{O}(m_h m_x n_h n_x)$ flops in general.

The fact above gives the worst-case cost of computing a convolution, but for many kernels of interest significantly more efficient algorithms exist. One especially nice case is when the matrix $\mathbf{H}$ has rank one (sometimes called a *separable filter*), in which case the rows and columns of the image can be convolved separately. To justify this, we make a few assertions but omit the details.

Fact 13.1.4 The convolution $\mathbf{h} * \mathbf{X}$ of a column vector and a matrix is the same as convolving $\mathbf{h}$ with each column of $\mathbf{X}$ separately. The convolution $\mathbf{h}^T * \mathbf{X}$ of a row vector and a matrix is the same as convolving $\mathbf{h}^T$ with each row of $\mathbf{X}$ separately.

Fact 13.1.5 If a matrix $\mathbf{H} = \mathbf{h}(\mathbf{h}')^T$ has rank 1, then $\mathbf{H} = \mathbf{h} * (\mathbf{h}')^T$.

Theorem 13.1.6 Convolution is associative: $\mathbf{A} * (\mathbf{B} * \mathbf{C}) = (\mathbf{A} * \mathbf{B}) * \mathbf{C}$.

Putting these together, we get the following theorem:

Theorem 13.1.7 If $\mathbf{H}$ has the rank 1 factorization $\mathbf{h}(\mathbf{h}')^T$, then $\mathbf{H} * \mathbf{X}$ can be computed by convolving the rows of $\mathbf{X}$ with $(\mathbf{h}')^T$ and the columns of $(\mathbf{h}')^T * \mathbf{X}$ with $\mathbf{h}$.

Proof. If $\mathbf{H} = \mathbf{h}(\mathbf{h}')^T = \mathbf{h} * (\mathbf{h}')^T$, then

$$\mathbf{H} * \mathbf{X} = (\mathbf{h} * (\mathbf{h}')^T) * \mathbf{X} = \mathbf{h} * ((\mathbf{h}')^T * \mathbf{X}).$$

■

We could equivalently convolve the columns first and the rows second. The key implication is that we can get major cost savings, similar to how matrix operations become cheaper when the matrix has a low-rank representation.

Fact 13.1.8 If $\mathbf{H}$ has rank one, then computing $\mathbf{H} * \mathbf{X}$ requires $\mathcal{O}(m_h n_x m_x) + \mathcal{O}(n_n n_x m_x)$ flops (assuming the dimensions of $\mathbf{H}$ are small compared to those of $\mathbf{X}$).

One convolution of particular interest is *Gaussian blur*, where the entries of $\mathbf{H}$ are weighted according to a normal distribution. In one variable, the PDF of a normal variable with standard deviation σ is given by (recall Definition 4.3.5)

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/(2\sigma^2)}.$$

In two dimensions, the PDF is given by

$$p(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} = \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/(2\sigma^2)}\right)\left(\frac{1}{\sigma\sqrt{2\pi}} e^{-y^2/(2\sigma^2)}\right) = p(x)p(y).$$

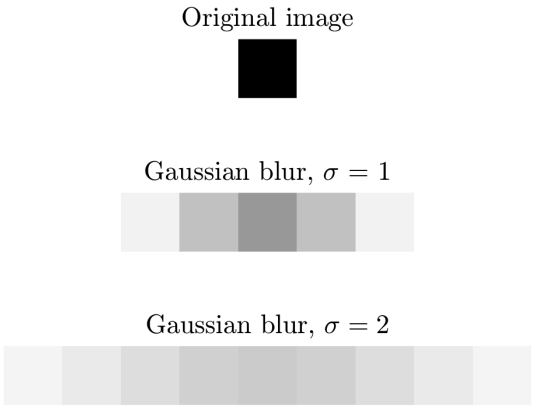


Figure 13.3: Gaussian blur in one dimension. Standard deviation σ is in number of pixels

Figure 13.4: Gaussian blur in two dimensions. Standard deviation σ is in number of pixels.

Fact 13.1.9 Gaussian blur is separable.

For image processing we use discrete approximations to the continuous distribution, such as the one in Example 13.4. Due to the rapid rate of decay of the tails of the Gaussian distribution, it is common to set values to zero if they fall more than three standard deviations away. Figures 13.3 and 13.4 show the effect of blurring on non-noisy images in one and two dimensions, respectively.

It is worth considering some of the properties of Gaussian blur as a linear transformation. In the idealized case (ignoring potential issues with boundary conditions), a vector all of whose entries are equal will not be affected by Gaussian blur. This means that the all-ones vector is an *eigenvector with eigenvalue 1*. In fact, it is (up to scaling) the only vector with eigenvalue one, and all other eigenvalues have magnitude less than 1. The other eigenvectors roughly correspond to sine waves of varying frequencies, where higher-frequency components are associated with smaller eigenvalues and therefore damped the most by blurring. You can more about this phenomenon in a class that deals with Fourier analysis, but the rough idea is that Gaussian blurring largely preserves low-frequency oscillations while reducing high-frequency ones (see Figure 13.5)

Fact 13.1.10 Gaussian noise has roughly equal components across all frequencies.

13.1.3 Truncated SVD

From the point of view of its eigenvalues and eigenvectors, Gaussian blur acts a little bit like a soft version of a projection: it attempts to preserve the parts of the image that we care about while

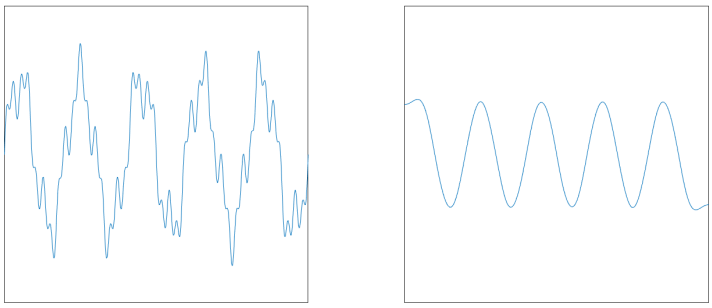


Figure 13.5: Gaussian blur largely preserves low-frequency components while damping high-frequency components. Left: original image. Right: after blurring.

eliminating the rest. We could also try to use the truncated SVD directly as a denoising tool directly. The idea is that if the amount of noise is not too large and if $\mathbf{A}$ has a good low-rank approximation, then the singular values and vectors of $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{E}$ will give a good approximation to the singular values and vectors of $\mathbf{A}$. One well-known result in the case of symmetric matrices is known as Weyl's Theorem; we present here the consequences of that theorem for perturbations to the SVD.

Theorem 13.1.11 — Weyl. Let $\mathbf{A}$ and $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{E}$ be arbitrary matrices (of the same size) with respective singular values $\sigma_1 \geq \cdots \geq \sigma_n$ and $\tilde{\sigma}_1 \geq \cdots \geq \tilde{\sigma}_n$. Then $|\sigma_i - \tilde{\sigma}_i| \leq \|\mathbf{E}\|_2$ for each $1 \leq i \leq n$.

In other words, if $\|\mathbf{E}\|_2$ is small then the singular values of $\widetilde{\mathbf{A}}$ are necessarily close to those of $\mathbf{A}$, and the relative approximation will be best for the largest singular values of $\mathbf{A}$. Other works provide bounds on the difference between the singular vectors of $\widetilde{\mathbf{A}}$ and $\mathbf{A}$, but these are more complicated since singular vectors are more sensitive to perturbations when the singular values are not well-separated.

■ **Example 13.5** Consider the original and perturbed matrices

$$\mathbf{A} = \begin{bmatrix} 1.01 & 0 \\ 0 & 1 \end{bmatrix}, \quad \widetilde{\mathbf{A}} = \begin{bmatrix} 1.01 & 0.01 \\ 0.01 & 1 \end{bmatrix}.$$

The perturbation satisfies $\|\mathbf{E}\|_2 = 0.01$, and the singular values are not changed much :

$$\sigma_1 = 1.01, \quad \sigma_2 = 1, \quad \tilde{\sigma}_1 \approx 1.0162, \quad \tilde{\sigma}_2 \approx 0.9938.$$

On the other hand, the right singular vectors are changed quite a bit:

$$\mathbf{V} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \widetilde{\mathbf{V}} \approx \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix}.$$

■

In any case, the hope is that if $\mathbf{A}$ has a good low-rank approximation, then the part of $\widetilde{A}$ corresponding to the top singular subspace will "mostly" consist of the information in $\mathbf{A}$, and that the part of $\widetilde{A}$ corresponding to the remainder will "mostly" consist of the noise from $\mathbf{E}$. So we hope to preserve the original information and eliminate the noise. In symbolic terms, if we fix a rank k and compute the truncated SVD $\widetilde{\mathbf{A}} = \widetilde{\mathbf{U}}_k \widetilde{\Sigma}_k \widetilde{\mathbf{V}}_k$, then ideally we will have

$$\Pi_{\widetilde{\mathbf{U}}_k} \widetilde{\mathbf{A}} = \Pi_{\widetilde{\mathbf{U}}_k} (\mathbf{A} + \mathbf{E}) \approx \Pi_{\widetilde{\mathbf{U}}_k} \mathbf{A} \approx \Pi_{\mathbf{U}_k} \mathbf{A}.$$

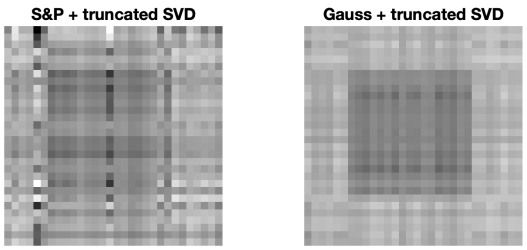


Figure 13.6: Truncated SVD as a tool for denoising. Rank 2 reconstructions shown.

Figure 13.7: Panel from PhD Comics ©2009, Jorge Cham. [3]

The original square image in Figure 13.1 is exactly rank 2, and Figure 13.6 shows the results of taking the rank-2 truncated SVD. The strategy is far more successful on Gaussian noise (which should be roughly evenly distributed across the singular subspaces of the image matrix) than on salt-and-pepper noise.

Two potential issues with this approach should be noted. First, we will not in general know how to set the truncated rank k in advance. Second, especially if we are concerned with image data, the "optimal" 2-norm or Frobenius norm approximations guaranteed by the Eckart-Young thoerem may not correspond to what *looks* best to the human eye. Gaussian blurring and median filtering are much more localized operations, and so may well look better according to the "eyeball" norm.

13.2 Deblurring

The previous section suggested that if an image has been corrupted by noise, we can use blurring to reduce the level of noise. A related question is, if an image has been blurred, can we expect to deblur it and accurately recover the original (e.g., a crime drama where a detective repeatedly yells "Enhance!" at grainy surveillance footage)? More generally, if we have access to noisy data of the form

$$\tilde{\mathbf{b}} = \mathbf{A}\mathbf{x} + \mathbf{e},$$

where $\mathbf{e}$ is "not too large" compared to $\mathbf{b} = \mathbf{A}\mathbf{x}$, how accurately can we expect to recover $\mathbf{x}$? Applications of this problem include medical imaging and determining the temperature of a material at an earlier point in time given later data. Depending on the situation we might be able to make some assumptions about the problem: for example, we might be able to reasonably model $\mathbf{e}$ as zero-mean Gaussian noise.

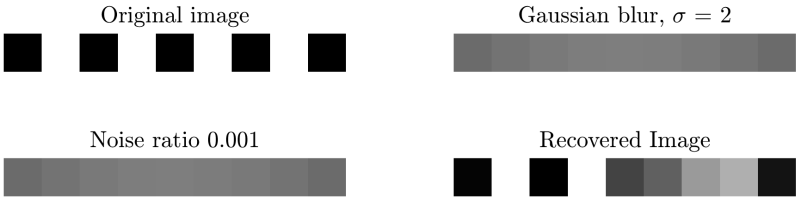


Figure 13.8: Deblurring is an ill-conditioned problem. Applying the exact inverse transformation can yield inaccurate results in the presence of a small amount of noise.

If $\mathbf{A}$ is well-conditioned (recall Sections 9.3 and 10.4.1), then there is no problem. We find directly from the definition of $\kappa(\mathbf{A})$ that

$$\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq \kappa(\mathbf{A})\frac{\|\tilde{\mathbf{b}}-\mathbf{b}\|_2}{\|\mathbf{b}\|_2} = \kappa(\mathbf{A})\frac{\|\mathbf{e}\|_2}{\|\mathbf{b}\|_2},$$

so if $\kappa(\mathbf{A})$ is close to 1 and $\|\mathbf{e}\|_2 \ll \|\mathbf{b}\|_2$, then we can expect to recover the true solution $\mathbf{x}$ to a reasonable degree of accuracy.

Fortunately for us, Gaussian blur is invertible. Unfortunately for us, it is not well-conditioned. The fact that blurring greatly dampens high-frequency oscillations implies that its smallest singular values can be quite close to zero. Figure 13.8 shows an example in one dimension, where blurring a rapidly oscillating image returns a vector with near-constant entries. The matrix representation of the blurring operation is only 9×9, but has a condition number of about $3.8 \cdot 10^3$. As a result, adding a small error at the level $\|\mathbf{e}\|_2/\|\mathbf{b}\|_2 = 0.001$ can result in the recovered solution being obviously flawed.

Results are even more pronounced in the 2-dimensional example in Figure 13.9, where $\|\mathbf{E}\|_F/\|\mathbf{B}\|_F \approx 0.03$. When the blurred image has no noise at all, accurate recovery is possible because the condition number (closer to 10^9 for this problem), while large compared to realistic levels of measurement error, is still small compared to the reciprocal of the unit roundoff $\mathrm{u} \approx 1.11 \cdot 10^{-16}$. In the presence of a small amount of Gaussian noise, however, our strategy fails.

13.2.1 Regularization by Truncated SVD

One potential strategy to overcome this challenge is to use *regularization*: roughly, replacing our ill-conditioned problem with a more well-conditioned one that we hope will have a similar solution. To do this, we find a low-rank approximation to $\mathbf{A}$ and use the pseudoinverse of the approximation in place of the inverse of $\mathbf{A}$.

Technique 13.2.1 — Truncated SVD. To find an approximate solution to the problem $\mathbf{A}\mathbf{x} = \tilde{\mathbf{b}}$, pick a target rank k and find the optimal rank-k approximation to $\mathbf{A}$,

$$\mathbf{A}_k = \mathbf{U}_k\Sigma_k\mathbf{V}_k^T.$$

Then use the Moore-Penrose inverse to find an approximate solution,

$$\tilde{\mathbf{x}} = \mathbf{A}_k^+\tilde{\mathbf{b}} = \mathbf{V}_k\Sigma_k^{-1}\mathbf{U}_k^T\tilde{\mathbf{b}}.$$

The rough idea is that since $\|\mathbf{A}_k^+\|_2 = \sigma_k^{-1}$, the condition number of this problem is σ_1/σ_k, which is potentially much smaller than σ_1/σ_n. If the solution $\mathbf{x}$ is mostly aligned with the dominant singular subspace of $\mathbf{A}$, then we can hope that not too much information (but lots of noise) will be discarded by taking the truncation.

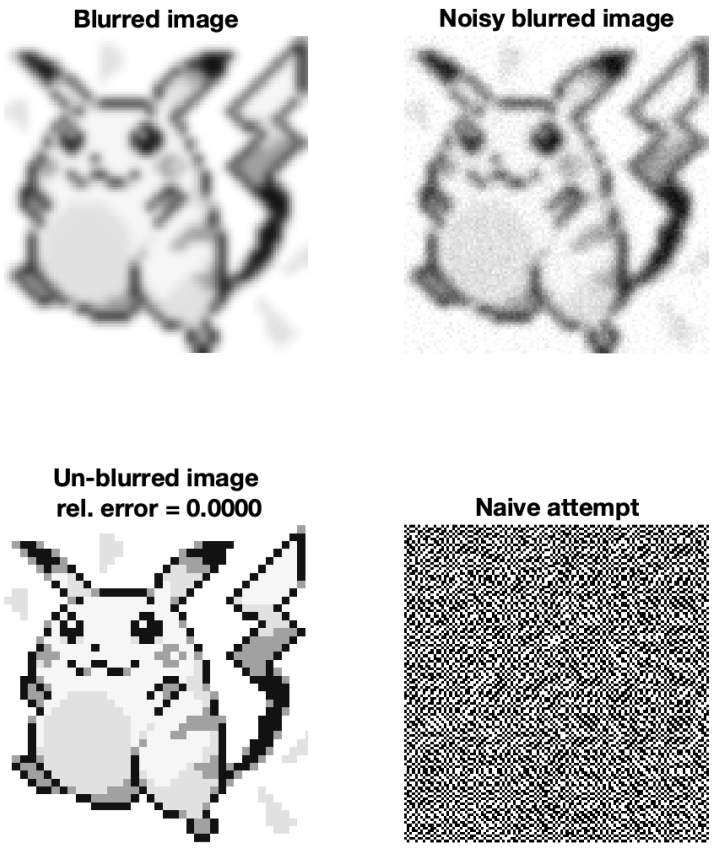


Figure 13.9: Without any noise at all, accurate recovery is possible. With minor noise, the straightforward strategy fails entirely.

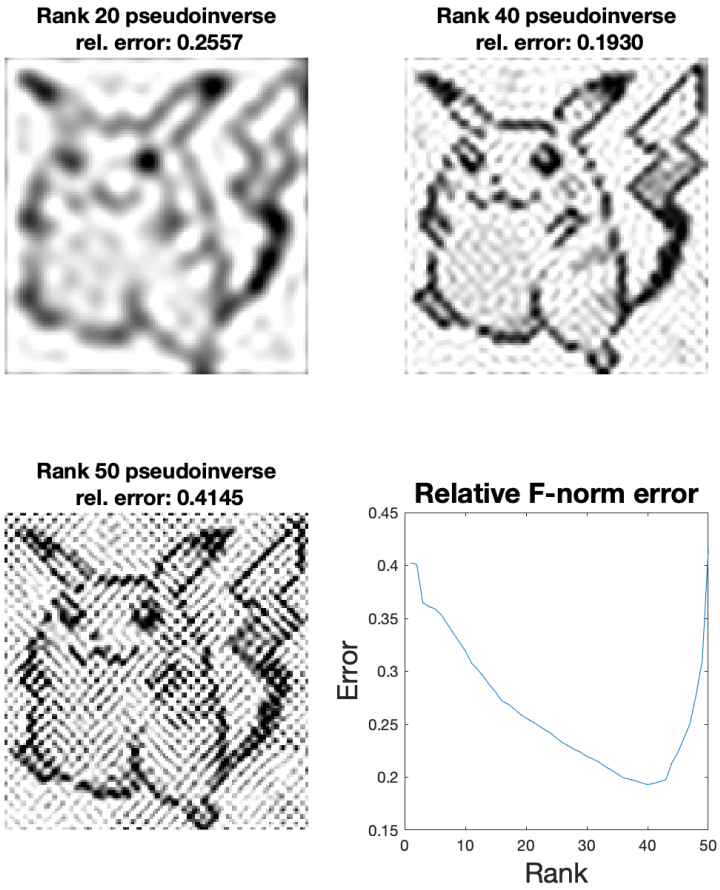


Figure 13.10: Errors in recovered solutions for various truncated SVD ranks.

More formally, we may say that $\mathbf{b}$ and $\mathbf{e}$ have the representations in terms of the left singular vectors of $\mathbf{A}$,

$$\mathbf{b} = \sum_{i=1}^{n} \beta_i \mathbf{u}_i, \quad \mathbf{e} = \sum_{i=1}^{n} \gamma_i \mathbf{u}_i.$$

Then the ideal and approximate solutions are

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \sum_{i=1}^{n} \frac{\beta_i}{\sigma_i}\mathbf{v}_i, \quad \tilde{\mathbf{x}}_k = \mathbf{A}_k^{+}\mathbf{b} = \sum_{i=1}^{k} \frac{\beta_i + \gamma_i}{\sigma_i}\mathbf{v}_i,$$

with 2-norm error

$$\|\mathbf{x} - \tilde{\mathbf{x}}_k\|_2 = \left\| \sum_{i=1}^{k} \frac{\gamma_i}{\sigma_i}\mathbf{v}_i + \sum_{i=k+1}^{n} \frac{\beta_i}{\sigma_i}\mathbf{v}_i \right\|_2 = \left(\sum_{i=1}^{k} \frac{\gamma_i^2}{\sigma_i^2} + \sum_{i=k+1}^{n} \frac{\beta_i^2}{\sigma_i^2} \right)^{1/2}.$$

Ideally, the coefficients $\beta_1, \ldots, \beta_k$ account for most of the mass of $\mathbf{b}$. In practice, knowing how to set k can be a tricky issue. Make it too large and the noise will be amplified; make it too small and not enough of the signal will be recovered. Figure 13.10 shows the effects of using various truncation ranks to deblur a 2D image.

13.2.2 L2 Regularization

A significant downside of the previous method is that computing the SVD is expensive. One alternate strategy is to use an approximate truncated SVD, which can be significantly cheaper to compute (see Section 10.5). Another strategy is to instead use L_2 *regularization*, also known as *ridge regression* or Tikhonov regularization. It gets its name from the L_2 norm, a.k.a. our good old Euclidean norm $\|\cdot\|_2$.

Technique 13.2.2 — L2 Regularization (ridge regression). To find an approximate solution to the problem $\mathbf{Ax} = \tilde{\mathbf{b}}$, pick a real number λ and instead solve

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2^2 + \lambda^2 \|\mathbf{x}\|_2^2. \tag{13.1}$$

One interpretation is that if the problem we face is that the small singular values of $\mathbf{A}$ greatly amplify the noise, then we can dampen this effect by adding a penalty term that forces the entries of $\mathbf{x}$ toward zero. When we try to solve the minimization problem in (13.1), the first term $\|\mathbf{Ax} - \mathbf{b}\|_2^2$ pressures us to make sure that the residual is small, and the second term $\lambda^2\|\mathbf{x}\|_2^2$ pressures us to ensure that the entries of $\mathbf{x}$ are small. Larger values of λ make the second term increasingly important. This formulation of the problem is attractive because it can be reorganized in the form

$$\min_{\mathbf{x}} \left\| \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} \right\|_2^2,$$

or in terms of the normal equations,

$$\tilde{\mathbf{x}}_\lambda = (\mathbf{A}^T\mathbf{A} + \lambda^2\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}.$$

If $\mathbf{A}$ has the SVD $\mathbf{U}\Sigma\mathbf{V}^T$, then it follows that

$$\begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix}^T \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix} = \mathbf{A}^T\mathbf{A} + \lambda^2\mathbf{I} = \mathbf{V}(\Sigma^2 + \lambda^2\mathbf{I})\mathbf{V}^T.$$

By taking square roots of the singular values, we get the following result.

Fact 13.2.3 If a matrix $\mathbf{A}$ has singular values $\sigma_1 \geq \cdots \geq \sigma_n$, then the augmented matrix $\mathbf{A}_\lambda := \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix}$ has singular values

$$\sqrt{\sigma_1^2 + \lambda^2} \geq \cdots \geq \sqrt{\sigma_n^2 + \lambda^2}$$

and condition number $\kappa(\mathbf{A}_\lambda) = \frac{\sqrt{\sigma_1^2+\lambda^2}}{\sqrt{\sigma_n^2+\lambda^2}}$.

So we end up with an effect rather similar to that of taking the truncated SVD: if $\sigma_1 > \lambda > \sigma_n$, then we get an approximate condition number $\kappa(\mathbf{A}_\lambda) \approx \sigma_1/\lambda$, which is ideally much smaller than $\sigma_1/\sigma_n = \kappa(\mathbf{A})$. As $\lambda \to \infty$ we would have $\kappa(\mathbf{A}_\lambda) \to 1$, meaning that the problem is increasingly insensitive to noise. Unfortunately, it would also follow that $\mathbf{x} \to \mathbf{0}$, so the result that we get would not be particularly useful. Once again, setting λ properly can be something of a delicate matter. Figure 13.11 shows the results of using various values for λ on the same image as before.

This type of regularization is also sometimes used when solving least squares problems $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2$ for which the matrix $\mathbf{A}$ is ill-conditioned. In the statistical setting, this might happen when two or more of the input variables are strongly correlated. In such a situation, the computed regression coefficients $\tilde{\mathbf{x}}$ can be highly sensitive to noise in the measurments.

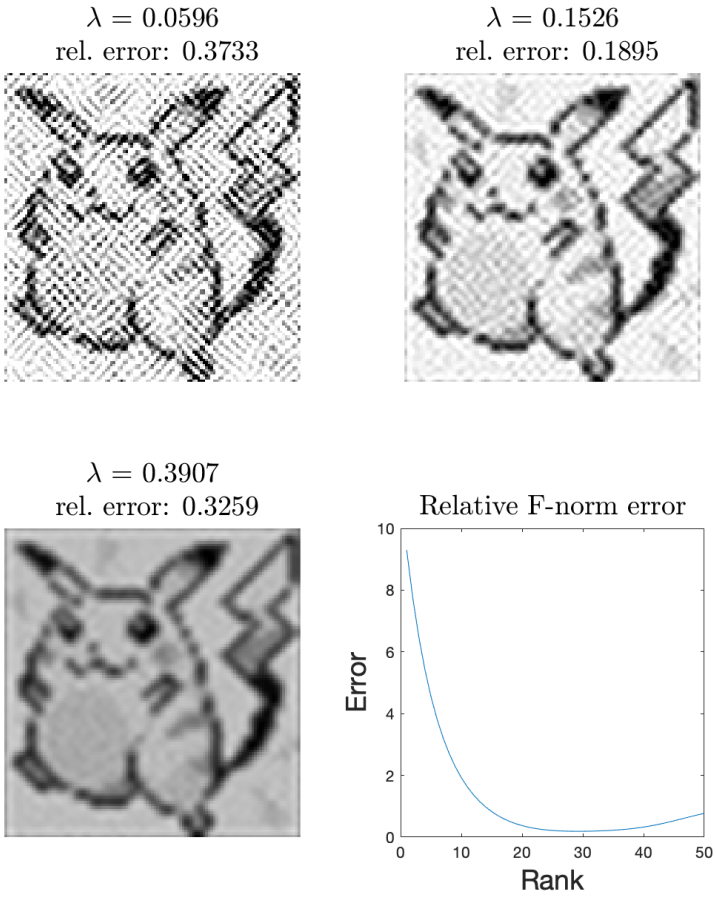


Figure 13.11: Errors in recovered solutions for L_2 regularization .

■ **Example 13.6** Come up with a linear model that attempts to predict the length of a person's right thumb in terms of the lengths of their other nine fingers. The matrix $\mathbf{A}$ will likely be close to having rank 1, but intuitively a "sensible" answer should probably involve all of the coefficients of $\mathbf{x}$ being greater than zero but not much larger than 1. Simply solving the least squares problem with the ill-conditioned matrix $\mathbf{A}$ might yield a result $\mathbf{x}$ with some very large positive coefficients and some very large negative coefficients. ■

13.2.3 LASSO

In some situations, we might want or expect our solution $\mathbf{x}$ to be *sparse*, or having most of its entries equal to zero. If we were to encode white pixels as zero and black as one, for example, then the Pikachu image being used as an example through this chapter would be fairly sparse. Here, we might consider using a method known as LASSO, or L_1 regularization, an acronym for *Least Absolute Shrinkage and Selection Operator*. It is similar to L_2 regularization, but uses the L_1 norm $\|\cdot\|_1$ instead.

Technique 13.2.4 — L1 Regularization (LASSO). To find an approximate solution to the problem $\mathbf{Ax} = \tilde{\mathbf{b}}$, pick a real number λ and instead solve

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1.$$

This problem can be significantly more expensive to solve because $\|\mathbf{x}\|_1$ is not a differentiable function of $\mathbf{x}$ (the gradient is undefined whenever a coordinate of $\mathbf{x}$ is equal to zero). The advantage over using the 2-norm is that having a 1-norm penalty pressures the coordinates of $\mathbf{x}$ to not just be *small*, but *equal to zero*. Depending on the application, we might consider the resulting solution to be of significantly higher quality.

We could also consider a compromise between L2 and L1 regularization, known as *elastic net regularization*. Here, we solve

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2^2 + \lambda_2 \|\mathbf{x}\|_2^2 + \lambda_1 \|\mathbf{x}\|_1,$$

which encourages the solution $\mathbf{x}$ to be sparse but also is guaranteed to have a unique minimum (LASSO sometimes does not).

■ **Example 13.7** Consider the system $\mathbf{Ax} = \mathbf{b}$, where

$$\mathbf{A} = \begin{bmatrix} 1.01 & 1 \\ 1 & 1.01 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

We present four solutions:

- The exact solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$;
- The rank-1 pseudoinverse solution $\mathbf{x} = \mathbf{A}_1^{+}\mathbf{b}$;
- Using L_2 regularization with $\lambda = 0.15$;
- Using L_1 regularization with $\lambda = 0.15$.

In order, the solutions are

$$\mathbf{x} \approx \begin{bmatrix} 101.0 \\ -99.00 \end{bmatrix}, \quad \begin{bmatrix} 0.995 \\ 0.995 \end{bmatrix}, \quad \begin{bmatrix} 1.432 \\ 0.547 \end{bmatrix}, \quad \begin{bmatrix} 1.953 \\ 0 \end{bmatrix}.$$

■

13.3 Matlab Commands

Filtering

- `imnoise(X,TYPE)`: Adds noise of various types to an image `X`.
- `medfilt2(X)`: Applies median filter to 2-dimensional image `X`.
- `conv(x,y)`: Convolves vectors `x` and `y`. See also `filter`.
- `conv2(X,Y)`: Performs 2-D convolution of matrices `X` and `Y`.

Regularization

- `ridge(Y,X,K)`: ridge or L_2 regression on (`X`,`Y`) data using parameter $\lambda = K$.
- `lasso(X,y)`: LASSO or L_1 regularization on (`X`,`y`) data. Returns multiple answers using a variety of different regularization coeffients λ. Other optional parameters can generalize to elastic net regularization.

IV Part Four

14. Covariance

We discussed Principal Component Analysis in Chapter 11 and least squares problems in Chapter 12 from the point of view of the singular value decomposition, but before we get to the topic of machine learning it's worth revisiting these concepts from the point of view of random variables. So let's start by introducing some new terminology.

14.1 Definition and Properties

Definition 14.1.1 — Covariance. The *covariance* of two random variables X and Y is defined as

$$\text{Cov}(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

Definition 14.1.2 If $\text{Cov}(X,Y) = 0$, then X and Y are *uncorrelated*. If $\text{Cov}(X,Y) > 0$, then X and Y are *positively correlated*. If $\text{Cov}(X,Y) < 0$, then X and Y are *negatively correlated*.

Covariance is one measure of the strength of the linear relationship between X and Y. For example, if two variables are positively correlated then the best-fit line predicting Y from X will have a positive slope. We note in order to compute the covariance of two random variables, we need to know the joint probability distribution (see Section 4.5).

Fact 14.1.1 For any random variable X and constant a, $\text{Cov}(a,X) = \text{Cov}(X,a) = 0$.

It is straightforward to verify from the definition of variance (Definition 5.2.2) that the variance of a random variable is its covariance with itself.

Fact 14.1.2 For any random variable X, $\mathbb{V}(X) = \text{Cov}(X,X)$.

As with the variance of a random variable, we can derive a handy alternate expression for the covariance of two random variables.

Theorem 14.1.3 For random variables X and Y,

$$\text{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].$$

Proof. To simplify the notation, let $\mu_X = \mathbb{E}[X]$ and $\mu_Y = \mathbb{E}[Y]$. Then

$$\begin{aligned}\text{Cov}(X,Y) &= \mathbb{E}[(X-\mu_X)(Y-\mu_Y)] \\ &= \mathbb{E}[XY - \mu_X Y - \mu_Y X + \mu_X\mu_Y] \\ &= \mathbb{E}[XY] - \mu_X\mathbb{E}[Y] - \mu_Y\mathbb{E}[X] + \mu_X\mu_Y \\ &= \mathbb{E}[XY] - \mu_X\mu_Y.\end{aligned}$$

■

Perhaps the most important feature of covariance is the close relation it has with inner products.

Theorem 14.1.4 The covariance function satisfies the following properties for real random variables.

1. (Positive Definiteness) If $\text{Cov}(X,X) = 0$, then X is constant with probability 1.
2. (Symmetry) For any X and Y, $\text{Cov}(X,Y) = \text{Cov}(Y,X)$.
3. (Bilinearity) For constants a and b and random variables X, Y, Z,

$$\text{Cov}(X, aY + bZ) = a\,\text{Cov}(X,Y) + b\,\text{Cov}(X,Z).$$

As we will see later, we will consequently be able to apply some of our more powerful linear algebra tools to problems involving variance and covariance.

14.1.1 Correlation and Independence

We note that covariance specifically measures the *linear* relationship between random variables. On one hand, independent random variables are necessarily uncorrelated.

Theorem 14.1.5 If X and Y are independent, then $\text{Cov}(X,Y) = 0$.

Proof. We give a proof in the case where the joint PDF $f_{X,Y}$ is defined. By Fact 4.4.1 it holds that $f_{X,Y}(x,y) = f_X(x)f_Y(y)$ for all x and y, so

$$\begin{aligned}\mathbb{E}[XY] &= \int_{x=-\infty}^{\infty}\int_{y=-\infty}^{\infty} xy f_{X,Y}(x,y)\,dx\,dy \\ &= \int_{x=-\infty}^{\infty}\int_{y=-\infty}^{\infty} xy f_X(x)f_Y(y)\,dx\,dy \\ &= \left(\int_{x=-\infty}^{\infty} x f_X(x)\,dx\right)\left(\int_{y=-\infty}^{\infty} y f_Y(y)\,dy\right) \\ &= \mathbb{E}[X]\mathbb{E}[Y].\end{aligned}$$

Thus $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$, meaning by Theorem 14.1.3 that $\text{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = 0$. ■

Unfortunately, the converse is not true. Two variables can have a very strong nonlinear relationship, but be weakly correlated.

■ **Example 14.1** Let $X, Y \overset{i.i.d.}{\sim} \text{Normal}(0,1)$. Then X and Y are uncorrelated since they are independent. ■

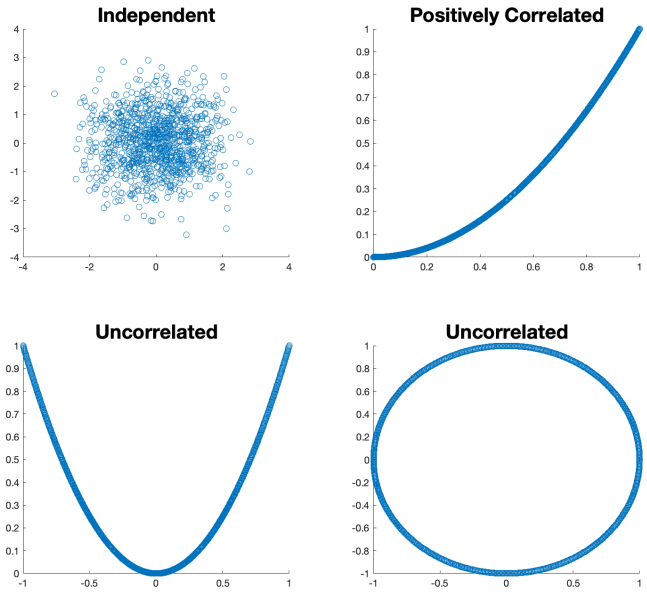


Figure 14.1: Samples drawn from distributions that are (top left) independent, (top right) positively correlated, and (bottom) uncorrelated but not independent.

■ **Example 14.2** Let $X \sim \text{Unif}(0,1)$ and let $Y = X^2$. Then X and Y are positively correlated since

$$\text{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[X^3] - \mathbb{E}[X]\mathbb{E}[X^2] = \frac{1}{4} - \left(\frac{1}{2}\right)\left(\frac{1}{3}\right) = \frac{1}{12} > 0.$$

■

■ **Example 14.3** Let $X \sim \text{Unif}(-1,1)$ and let $Y = X^2$. Then X and Y are uncorrelated since

$$\text{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[X^3] - \mathbb{E}[X]\mathbb{E}[X^2] = 0 - 0 \cdot \left(\frac{1}{3}\right) = 0,$$

but not independent since the value of Y can be predicted perfectly from the value of X. ■

■ **Example 14.4** Let $\Theta \sim \text{Unif}(0, 2\pi)$, and let $X = \cos\Theta$ and $Y = \sin\Theta$. Then X and Y are not independent since $X^2 + Y^2 = 1$, but

$$\text{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[\cos\Theta\sin\Theta] - \mathbb{E}[\cos\Theta]\mathbb{E}[\sin\Theta] = 0 - 0 \cdot 0 = 0.$$

To see that $\mathbb{E}[\cos\Theta\sin\Theta] = 0$, use the identity $\sin 2\theta = 2\sin\theta\cos\theta$. ■

Figure 14.1 shows some plots of samples drawn from the distributions in each of these examples. It is worth noting that although the bottom two plots are both symmetric about the y-axis, this is not necessary for the underlying variables X and Y to be uncorrelated. We present two more examples without this type of symmetry, illustrated in Figure 14.2.

■ **Example 14.5** Let $U_1, U_2 \overset{i.i.d.}{\sim} \text{Unif}(0,1)$, and let $X = 2U_1 + U_2$ and $Y = U_1 - 2U_2$. Then X and Y are not independent, since (for example) if $X = 3$ then $U_1 = U_2 = 1$, so $Y = -1$. On the other

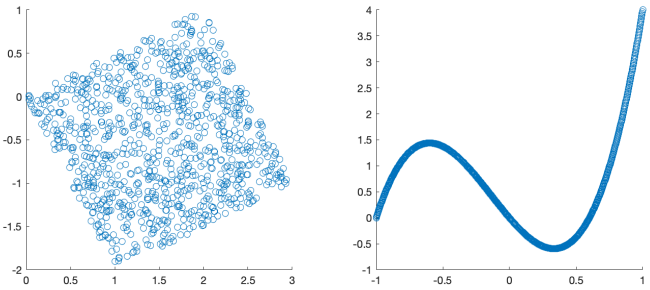


Figure 14.2: Samples drawn from two more distributions that are uncorrelated but not independent.

hand,

$$\begin{aligned}\text{Cov}(X,Y) &= \text{Cov}(2U_1+U_2, U_1-2U_2)\\ &= 2\text{Cov}(U_1,U_1)-4\text{Cov}(U_1,U_2)+\text{Cov}(U_2,U_1)-2\text{Cov}(U_2,U_2)\\ &= 2\text{Cov}(U_1,U_1)-2\text{Cov}(U_2,U_2)\\ &= 0,\end{aligned}$$

since U_1 and U_2 are independent but identically distributed. ■

■ **Example 14.6** Let $X \sim \text{Unif}(-1,1)$ and let $Y = 5X^3+2X^2-3X$. Then Y is a function of X, so the two are clearly not independent, but

$$\begin{aligned}\text{Cov}(X,Y) &= \mathbb{E}[XY]-\mathbb{E}[X]\mathbb{E}[Y]\\ &= \mathbb{E}[5X^4+2X^3-3X^2]-0\cdot\mathbb{E}[Y]\\ &= 5\mathbb{E}[X^4]+2\mathbb{E}[X^3]-3\mathbb{E}[X^2]\\ &= 5(1/5)+2\cdot 0-3(1/3)\\ &= 0.\end{aligned}$$

■

14.1.2 Correlation and Causation

You have perhaps heard the cliché that "correlation does not imply causation". It's an oft-repeated phrase, but certainly a true one. Some correlations are simply coincidence: it is observed at [24] for example, that between the years 1999 and 2009 the number of people who drowned by falling into a pool is positively correlated with the number of movies Nicholas Cage has appeared in. Other times, there might be a hidden underlying factor. Ice cream sales are known to be correlated with property crime over time, but this doesn't mean that eating ice cream gives you criminal tendencies and it doesn't mean that burglars celebrate a heist by going out for ice cream. Both are seasonal variables that peak in the summer.

Less commonly observed is the fact that causation does not neccessarily imply correlation! The economist Rachael Meager [16] poses an example:

> Imagine driving a car, reaching a hill and pumping the gas as you begin to go up so that your speed is constant. The correlation between pressing on the gas and the speed of the car is zero but they're obviously causally related, it's that the agent is optimizing speed!

In general, determining cause and effect is a difficult matter, not that such difficulty will stop statisticians, social scientists, economists, or medical researchers from trying. Causation is also not a concept with a good purely mathematical formulation, although some have tried to frame it in terms of probability theory, saying that a cause "raises the probability" of its effect. Such formulations have faced criticism in turn, since the very phrase "raises the probability" implies the notion of a counterfactual. Modern probability theory, grounded in the axioms presented in Definition 4.1.3, concerns itself with sets and functions and does not traditionally deal with counterfactuals. Consequently, we will not discuss causal inference further.

14.2 Correlation and Least Squares

The covariance doesn't immediately tell use how closely related two random variables X and Y are because it is sensitive to scaling. Suppose we want to examine the relation between average global temperature Y and the year X, for example. If we switch the units of Y from Celsius to Fahrenheit, then $\text{Cov}(X,Y)$ will change by a factor of $9/5$ even though the data represents the same underlying relationship. Accordingly, it is reasonable to want a measure that does not depend on the variances of the individual variables.

Definition 14.2.1 — Correlation. Given two random variables X and Y with positive variance, the *correlation* of X and Y is

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma(X)\sigma(Y)}.$$

Here the relation between covariance and inner products comes into play. In Theorem 8.3.13, it was mentioned that the angle between two vectors can be defined as

$$\theta(\mathbf{x},\mathbf{y}) = \cos^{-1}\left(\frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2}\right).$$

The Cauchy-Schwarz inequality (Theorem 3.4.3) guarantees that

$$-1 \leq \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2} \leq 1$$

for all nonzero vectors $\mathbf{x}$ and $\mathbf{y}$, and the covariance of two random vectors is similarly bound.

Theorem 14.2.1 For any random vectors X and Y, $|\text{Cov}(X,Y)| \leq \sigma(X)\sigma(Y)$.

Corollary 14.2.2 For any random vectors X and Y with positive variance, $-1 \leq \rho_{X,Y} \leq 1$.

In this sense, we can think of the correlation as the cosine of the "angle" between two random variables. Positively correlated variables have an acute angle, negatively correlated ones have an obtuse angle, and uncorrelated variables are orthogonal.

14.2.1 Least Squares, Revisited

Let's take a second look at the least squares problem, this time in terms of random variables. For random variables X and Y with $\mathbb{V}(X) > 0$, we want to predict Y as a linear function of X,

$$Y = \alpha X + \beta + R, \tag{14.1}$$

where R represents the residual. We will give ourselves the goal of minimizing the *mean squared error* (MSE), or $\mathbb{E}[R^2]$. A useful little theorem allows us to express the MSE in terms of the *bias* of the estimator and the *variance* of the residuals.

Theorem 14.2.3 — Bias-Variance Tradeoff. For any random variable R,

$$\mathbb{E}[R^2] = \mathbb{E}[R]^2 + \mathbb{V}(R).$$

Proof. The claim follows immediately from the formula $\mathbb{V}(R) = \mathbb{E}[R^2] - \mathbb{E}[R]^2$. ■

Definition 14.2.2 If Equation 14.1 satisfies $\mathbb{E}[R] = 0$, then the estimator $Y \approx \alpha X + \beta$ is *unbiased*.

Before using this expression to tackle the more general linear model, we first give a smaller result to parallel that of Fact 12.4.1.

Fact 14.2.4 For the constant model $Y = \beta + R$ the MSE is minimized when $\beta = \mathbb{E}[Y]$.

Proof. Using Theorem 14.2.3, we write the MSE as

$$\mathbb{E}[R^2] = \mathbb{E}[R]^2 + \mathbb{V}(R) = \mathbb{E}[Y-\beta]^2 + \mathbb{V}(Y-\beta) = (\mathbb{E}[Y]-\beta)^2 + \mathbb{V}(Y).$$

The bias is zero when $\beta = \mathbb{E}[Y]$, and the variance is constant for all β. ■

Now for the more general model.

Theorem 14.2.5 The MSE estimator for the model $Y = \alpha X + \beta + R$ is given by

$$\alpha = \rho_{X,Y}\frac{\sigma(Y)}{\sigma(X)},$$
$$\beta = \mathbb{E}[Y] - \alpha\mathbb{E}[X].$$

Proof. Using Theorem 14.2.3, we write the MSE as

$$\mathbb{E}[R^2] = \mathbb{E}[R]^2 + \mathbb{V}(R) = \mathbb{E}[Y-(\alpha X+\beta)]^2 + \mathbb{V}(Y-(\alpha X+\beta)).$$

Focusing first on minimizing the variance term, we observe that

$$\begin{aligned}\mathbb{V}(Y-(\alpha X+\beta)) &= \mathbb{V}(Y-\alpha X)\\ &= \mathrm{Cov}(Y-\alpha X, Y-\alpha X)\\ &= \mathbb{V}(Y) - 2\alpha\,\mathrm{Cov}(X,Y) + \alpha^2\mathbb{V}(X).\end{aligned}$$

Setting the derivative with respect to α to zero gives the solution

$$\alpha = \frac{\mathrm{Cov}(X,Y)}{\mathbb{V}(X)} = \frac{\mathrm{Cov}(X,Y)}{\sigma(X)\sigma(Y)}\cdot\frac{\sigma(Y)}{\sigma(X)} = \rho_{X,Y}\frac{\sigma(Y)}{\sigma(X)}.$$

Then by Fact 14.2.4, the bias term $\mathbb{E}[(Y-\alpha X)-\beta]^2$ is minimized (zero, in fact) when

$$\beta = \mathbb{E}[Y-\alpha X] = \mathbb{E}[Y] - \alpha\mathbb{E}[X].$$

■

From this theorem we make two observations. First, the slope α of the best fit line is equal to the correlation coefficient $\rho_{X,Y}$ precisely when X and Y have been normalized to have the same standard deviation. Second, the resulting linear estimator is unbiased, i.e., $\mathbb{E}[R] = 0$, and the best fit line passes through the center of mass $(\mathbb{E}[X], \mathbb{E}[Y])$.

R We could alternately prove the theorem by setting to zero the partial derivatives of the MSE with respect to α and β. We would get the same answer, but would find it via an expresion that closely resembles the normal equations.

14.2.2 Residual Error

Given our optimal values for α and β given by Theorem 14.2.5, we find that the resulting mean squared error is given by

$$\begin{aligned}\mathbb{E}[R^2] &= \mathbb{E}[R]^2 + \mathbb{V}(R) \\ &= 0 + \left[\mathbb{V}(Y) - 2\alpha\,\mathrm{Cov}(X,Y) + \alpha^2\mathbb{V}(X)\right] \\ &= \mathbb{V}(Y) - 2\frac{\mathrm{Cov}(X,Y)}{\mathbb{V}(X)}\cdot\mathrm{Cov}(X,Y) + \left(\frac{\mathrm{Cov}(X,Y)}{\mathbb{V}(X)}\right)^2\mathbb{V}(X) \\ &= \mathbb{V}(Y) - \frac{\mathrm{Cov}(X,Y)^2}{\mathbb{V}(X)} \\ &= (1-\rho_{X,Y}^2)\mathbb{V}(Y).\end{aligned}$$

Since $\mathbb{E}[R] = 0$, we also know that

$$\mathbb{V}(R) = \mathbb{E}[R^2] - \mathbb{E}[R]^2 = \mathbb{E}[R^2].$$

We can therefore deduce from the correlation coefficient how large the variance of the residual R is compared to the variance of Y.

Definition 14.2.3 — r^2 coefficient. Given variables X and Y, the *coefficient of determination*, or r^2 *coefficient*, is given by

$$r^2 = \rho_{X,Y}^2.$$

Given the best-fit coefficients to the linear model $Y = \alpha X + \beta + R$, it satisfies

$$\frac{\mathbb{V}(R)}{\mathbb{V}(Y)} = 1 - r^2.$$

Put another way, the r^2 coefficient is a measure of how much of the variance in Y can be "explained" by its linear relation with the variable X. If r^2 is near 1, then Y and X have a strong linear relationship. If r^2 is close to zero, then the line $\alpha X + \beta$ is not much better as a prediction of Y than just the constant guess $\mathbb{E}[Y]$.

■ **Example 14.7** Consider the random variables

$$\begin{aligned}X &\sim \mathrm{Unif}(0,5), \\ E &\sim \mathrm{Normal}(0,1), \\ Y &= 0.4X + 0.2E,\end{aligned}$$

where X and E are independent. The variables have variances

$$\mathbb{V}(X) = \frac{25}{12} \approx 2.08, \quad \mathbb{V}(E) = 1, \quad \mathbb{V}(Y) = 0.4^2\mathbb{V}(X) + 0.2^2\mathbb{V}(E) = \frac{28}{75} \approx 0.373.$$

The covariance of X and Y is given by

$$\mathrm{Cov}(X,Y) = \mathrm{Cov}(X, 0.4X + 0.2E) = 0.4\,\mathrm{Cov}(X,X) + 0.2\,\mathrm{Cov}(X,E) = 0.4\mathbb{V}(X) + 0 = \frac{5}{6},$$

and the correlation coefficient is therefore

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma(X)\sigma(Y)} = \frac{5}{2\sqrt{7}} \approx 0.945.$$

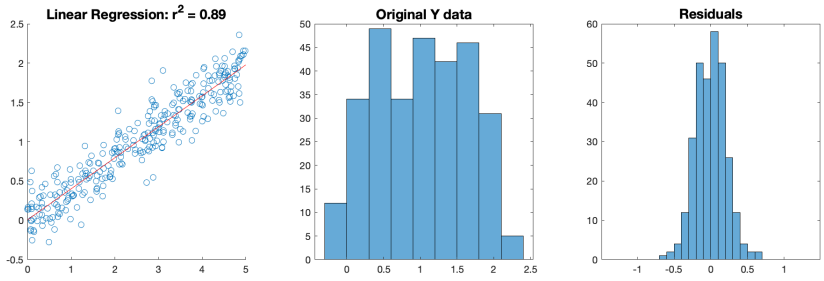


Figure 14.3: Samples from strongly correlated distributions. The residual variance is much smaller than the variance of the original y data.

The optimal linear fit is $Y = 0.4X$ with residual $0.2E$, and the r^2 coefficient is

$$r^2 = \rho_{X,Y}^2 = 1 - \frac{\mathbb{V}(0.2E)}{\mathbb{V}(Y)} = \frac{25}{28} \approx 0.893.$$

This indicates a pretty strong linear relation! Figure 14.3 shows an estimate of the best linear model given 300 samples from the above distributions. The estimated fit is

$$Y \approx 0.394X + 0.009,$$

with estimated coefficient of determination $r^2 \approx 0.89$. ∎

14.2.3 Application to Monte Carlo Integration

With covariance defined, we finally can give an exact answer to a problem regarding Monte Carlo integration from Section 6.3.1. Specifically, if we want to estimate $\mathbb{E}[X]$ by sampling X, and have access to a variable Y that "resembles" X and for which the expectation $\mathbb{E}[Y] = \mu_Y$ is known, then we can save time by instead sampling Y and X simultaneously and using the identity

$$\mathbb{E}[X] = \mathbb{E}[X - Y] + \mu_Y.$$

To be more precise: we want to find an Y that is *strongly correlated* with X, pick a coefficient α, and use the identity

$$\mathbb{E}[X] = \mathbb{E}[X - \alpha Y] + \alpha\mu_Y.$$

What value of α, then, is best? It's the one that minimizes $\mathbb{V}(X - \alpha Y)$, or exactly the coefficient from our best-fit linear model. The technique is summarized as follows:

Technique 14.2.6 — Sampling with Control Variates. In order to sample a variable X with control variate Y whose mean μ_Y is known,

1. Take a small pilot sample of X and Y;
2. Estimate $\mathbb{V}(Y)$ and $\mathrm{Cov}(X,Y)$;
3. Set $\alpha = \mathrm{Cov}(X,Y)/\mathbb{V}(Y)$;
4. Estimate $\mathbb{E}[X]$ by sampling $\mathbb{E}[X - \alpha Y] + \alpha\mu_Y$.

The stronger the correlation between X and Y, the bigger the variance reduction (and resulting cost savings) will be.

14.3 Covariance Matrices

In order to apply our linear algebra techniques more broadly, it is worth defining jointly distributed random variables in terms of vectors.

Definition 14.3.1 A *random vector* is a vector of random variables:

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}.$$

Expectations can be computed elementwise.

Definition 14.3.2 For a random vector $\mathbf{X}$, the expected value is defined as

$$\mathbb{E}[\mathbf{X}] = \begin{bmatrix} \mathbb{E}[X_1] \\ \mathbb{E}[X_2] \\ \vdots \\ \mathbb{E}[X_n] \end{bmatrix}$$

and will often be denoted by $\mu_{\mathbf{X}}$.

Variance is a little more complicated, since we would like to measure not just the variance of each variable but also the covariance between pairs of variables. Instead of a length-n vector, we define an $n \times n$ matrix.

Definition 14.3.3 The *covariance matrix* of a random vector $\mathbf{X}$ is given by

$$K_{\mathbf{XX}} = \mathbb{V}(\mathbf{X}) = \begin{bmatrix} \mathbb{V}(X_1) & \text{Cov}(X_1, X_2) & \cdots & \text{Cov}(X_1, X_n) \\ \text{Cov}(X_2, X_1) & \mathbb{V}(X_2) & \cdots & \text{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_n, X_1) & \text{Cov}(X_n, X_2) & \cdots & \mathbb{V}(X_n) \end{bmatrix}.$$

In general, the (i, j) entry of $K_{\mathbf{XX}}$ is $\text{Cov}(X_i, X_j)$.

Put compactly, we may say:

Fact 14.3.1 For a random vector $\mathbf{X}$,

$$K_{\mathbf{XX}} = \mathbb{E}\left[(\mathbf{X} - \mu_{\mathbf{X}})(\mathbf{X} - \mu_{\mathbf{X}})^T\right] = \mathbb{E}[\mathbf{X}\mathbf{X}^T] - \mu_{\mathbf{X}}\mu_{\mathbf{X}}^T.$$

The final inequality introduces an alternate expression for the covariance matrix, in a natural extension of Theorems 5.2.1 and 14.1.3. As with the previous unbiased estimate of variance (Definition 6.2.3) that employs Bessel's correction, we can normalize by $n-1$ rather than n to get an unbiased estimate of the covariance matrix from a collection of samples.

Fact 14.3.2 Given random vectors $\mathbf{X}_1, \ldots, \mathbf{X}_n \overset{i.i.d.}{\sim} \mathbf{X}$, the matrix

$$\mathbf{Q} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{X}_i - \overline{\mathbf{X}})(\mathbf{X}_i - \overline{\mathbf{X}})^T$$

is an unbiased estimator of $K_{\mathbf{XX}}$, where

$$\overline{\mathbf{X}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{X}_i$$

is the sample average of the random vectors.

Since $K_{\mathbf{XX}}$ is a matrix, we might inquire what sorts of properties it has that relate more closely to linear algebra.

Theorem 14.3.3 For any random vector $\mathbf{X}$, $K_{\mathbf{XX}}$ is symmetric positive semidefinite: that is, it satisfies

1. $K_{\mathbf{XX}} = K_{\mathbf{XX}}^T$;
2. $\mathbf{a}^T K_{\mathbf{XX}} \mathbf{a} \geq 0$ for any vector $\mathbf{a}$.
3. All eigenvalues of $K_{\mathbf{XX}}$ are nonnegative.

Proof. First, $K_{\mathbf{XX}}$ is symmetric since $\mathrm{Cov}(X,Y) = \mathrm{Cov}(Y,X)$ in general. Second, it is positive semidefinite since for any $\mathbf{a} = [a_1, \ldots, a_n]^T$ we find that

$$\begin{aligned}
\mathbf{a}^T K_{\mathbf{XX}} \mathbf{a} &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \mathrm{Cov}(X_i, X_j) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{Cov}(a_i X_i, a_j X_j) \\
&= \mathrm{Cov}\left(\sum_{i=1}^{n} a_i X_i, \sum_{j=1}^{n} a_j X_j\right) \\
&= \mathbb{V}\left(\mathbf{a}^T \mathbf{X}\right) \\
&\geq 0.
\end{aligned}$$

Finally, by the Spectral Theorem (Theorem 10.1.3), since $K_{\mathbf{XX}}$ is symmetric it has an eigendecomposition $K_{\mathbf{XX}} = \mathbf{Q}\Lambda\mathbf{Q}^T$. For any eigenvector $\mathbf{q}$, we know from the above that

$$0 \leq \mathbf{q}^T K_{\mathbf{XX}} \mathbf{q} = \lambda.$$

Thus all eigenvalues of $K_{\mathbf{XX}}$ are nonnegative. ■

Note that the proof implies more specifically that $\mathbb{V}(\mathbf{a}^T\mathbf{X}) = \mathbf{a}^T K_{\mathbf{XX}} \mathbf{a}$ for any vector $\mathbf{a}$. We can generalize this results to allow for more general linear and affine transformations on $\mathbf{X}$.

Theorem 14.3.4 For any random n-vector $\mathbf{X}$, constant matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, and constant vector $\mathbf{b} \in \mathbb{R}^m$,

$$\mathbb{V}(\mathbf{AX} + \mathbf{b}) = \mathbf{A}\mathbb{V}(\mathbf{X})\mathbf{A}^T = \mathbf{A}K_{\mathbf{XX}}\mathbf{A}^T.$$

Proof. Let $\mathbf{Y} = \mathbf{AX} + \mathbf{b}$. Then $\mu_{\mathbf{Y}} = \mathbf{A}\mu_{\mathbf{X}} + \mathbf{b}$ by the linearity of expectations, and so

$$\mathbf{Y} - \mu_{\mathbf{Y}} = (\mathbf{AX} + \mathbf{b}) - (\mathbf{A}\mu_{\mathbf{X}} + \mathbf{b}) = \mathbf{AX} - \mathbf{A}\mu_{\mathbf{X}} = \mathbf{A}(\mathbf{X} - \mu_{\mathbf{X}}).$$

Therefore by the compact expression for $K_{\mathbf{XX}}$ from Fact 14.3.1, we get that

$$\begin{aligned} K_{\mathbf{XX}} &= \mathbb{E}\left[(\mathbf{Y}-\mu_{\mathbf{Y}})(\mathbf{Y}-\mu_{\mathbf{Y}})^T\right] \\ &= \mathbb{E}\left[\mathbf{A}(\mathbf{X}-\mu_{\mathbf{X}})(\mathbf{X}-\mu_{\mathbf{X}})^T\mathbf{A}^T\right] \\ &= \mathbf{A}\mathbb{E}\left[(\mathbf{X}-\mu_{\mathbf{X}})(\mathbf{X}-\mu_{\mathbf{X}})^T\right]\mathbf{A}^T \\ &= \mathbf{A}K_{\mathbf{XX}}\mathbf{A}^T. \end{aligned}$$

■

■ **Example 14.8** If the entries of a random vector $\mathbf{X}$ are pairwise uncorrelated, then $K_{\mathbf{XX}} = \mathbf{I}_n$. ■

■ **Example 14.9** If all entries of $\mathbf{X}$ are identical, $\mathbf{X} = [X, X, \ldots, X]^T$, then the covariance matrix has rank one:

$$K_{\mathbf{XX}} = \begin{bmatrix} \mathbb{V}(X) & \mathbb{V}(X) & \cdots & \mathbb{V}(X) \\ \mathbb{V}(X) & \mathbb{V}(X) & \cdots & \mathbb{V}(X) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{V}(X) & \mathbb{V}(X) & \cdots & \mathbb{V}(X) \end{bmatrix} = \mathbb{V}(X)\mathbb{1}\mathbb{1}^T.$$

■

■ **Example 14.10** If the entries of $\mathbf{X}$ are rescaled to have unit standard deviation, then the entries of the covariance matrix will represent the correlations between pairs of variables, with all ones on the diagonal. That is, if we set

$$\mathbf{D} = \begin{bmatrix} \sigma(X_1) & & & \\ & \sigma(X_2) & & \\ & & \ddots & \\ & & & \sigma(X_n) \end{bmatrix}, \mathbf{Y} = \mathbf{D}^{-1}\mathbf{X},$$

then $\mathbf{K}_{\mathbf{YY}} = \mathbf{D}^{-1}K_{\mathbf{XX}}\mathbf{D}^{-1}$ has (i,j) entry equal to ρ_{X_i,X_j}. In Matlab, this matrix can be estimated from a sample with the command `corrcoef`. ■

14.3.1 Application to Portfolio Theory

One problem that an investor might face is how to allocate their money across a range of assets in a way that maximizes the expected rate of return and minimizes the risk. Some assets, such as US Treasury bonds, have a low rate of return but almost no risk of default. Others, such as fledgling startups, might have a higher rate of default but offer a greater rate of return. The performances of different companies are not in general independent: in particular, companies in the same industry may have highly correlated performances since they are affected similarly by economic or political events.

So say that we have a collection of n possible assests to invest in, whose rates of return are modeled as a random vector with expected value $\mathbf{m}$ and covariance matrix $\mathbf{K}$. Let the vector $\mathbf{x}$ represent the proportion of our funds to invest in each asset, so that $\sum_{i=1}^n \mathbf{x} = \mathbb{1}^T\mathbf{x} = 1$ (we will assume that the financial tools exist that allow the entries of $\mathbf{x}$ to take on negative values). Using variance as a proxy for risk, we could set a desired rate of return μ and get the problem

$$\min_{\mathbf{x}} \mathbf{x}^T\mathbf{K}\mathbf{x} \quad \text{subject to} \quad \mathbf{m}^T\mathbf{x} = \mu,\ \mathbb{1}^T\mathbf{x} = 1.$$

Using Lagrange multipliers tells us that any solution must satisfy

$$2K\mathbf{x} + \lambda_1\mathbf{m} + \lambda_2\mathbb{1} = \mathbf{0}$$

for some $\lambda_1, \lambda_2 \in \mathbb{R}$. Combining this constraint with the two linear requirements on $\mathbf{x}$ (desired rate of return; investments must sum to one), we find that the solution must satisfy the linear system

$$\begin{bmatrix} \mathbf{K} & \mathbf{m} & \mathbb{1} \\ \mathbf{m}^T & 0 & 0 \\ \mathbb{1}^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mu \\ 1 \end{bmatrix}.$$

More generally, we observe that diversifying investments can help reduce risk, but the benefit is lessened if the assets have strongly correlated performances. On the opposite end, if two assets are weakly or even negatively correlated, then investing in a mix of the two can greatly reduce the variance in the rate of return.

14.4 A Second Look at PCA

In Section 11.3.2 we defined PCA in termes of matrices, interpreting an $(n \times d)$ data matrix as a collection of n observations with d attributes each, or n points in d dimensions. If we assume that the n observations are i.i.d. samples from a fixed probability distribution, then PCA can give us information about the underlying properties of the distribution. Here, we rederive PCA from the point of view of the underlying distribution itself.

Theorem 14.4.1 — Principal Component Analysis. Let $\mathbf{X}$ be a random vector. Then there exists an orthogonal linear transformation $\mathbf{V}$ such that the components of $\mathbf{V}^T\mathbf{X}$ are uncorrelated.

Proof. Let $K_{\mathbf{XX}}$ be the covariance matrix of $\mathbf{X}$. By Theorem 14.3.3 it is symmetric with all nonnegative eigenvalues, so we can write it in the form

$$K_{\mathbf{XX}} = \mathbf{V}\Sigma^2\mathbf{V}^T$$

for some orthogonal matrix $\mathbf{V}$ and diagonal matrix Σ. Then by Theorem 14.3.4, if we set $\mathbf{Y} = \mathbf{V}^T\mathbf{X}$ it follows that

$$K_{\mathbf{YY}} = \mathbf{V}^T\mathbf{K}_{\mathbf{XX}}\mathbf{V} = \mathbf{V}^T(\mathbf{V}\Sigma^2\mathbf{V}^T)\mathbf{V} = \Sigma^2.$$

Since Σ^2 is diagonal, the components of $\mathbf{Y}$ are uncorrelated. ■

If we order the diagonal elements of Σ^2 from largest to smallest, the columns of $\mathbf{V}$ give us the linear combinations of elements in $\mathbf{X}$ that have the greatest variance down to the least. That is, for a random d-vector $\mathbf{X}$ the solution to the problem

$$\max_{\mathbf{a}\in\mathbb{R}^d} \mathbb{V}\left(\sum_{i=1}^{d} a_i X_i\right) \quad \text{subject to} \quad \|\mathbf{a}\|_2 = 1$$

is given by

$$Y_1 = \sum_{i=1}^{d} v_{i1} X_i,$$

with $\mathbb{V}(Y_1) = \sigma_1^2$.

14.5 Multivariate Normal Distributions

Now for one of the most important developments: extending the definition of a Gaussian distribution to higher dimensions.

Definition 14.5.1 — Gaussian Random Vector. Assuming $\mathbf{K} \in \mathbb{R}^{d\times d}$ is symmetric with all positive eigenvalues (i.e., positive definite), a *Gaussian random vector* $\mathbf{X} \sim \text{Normal}(\mu, \mathbf{K})$ has the joint probability density function

$$f_{\mathbf{X}}(x_1,\ldots,x_d) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{K})}} e^{-(\mathbf{x}-\mu)^T \mathbf{K}^{-1}(\mathbf{x}-\mu)/2}.$$

The PDF is called a *multivariate normal distribution*. As is required of PDFs in general, it integrates to 1 over $\mathbb{R}^d$.

The definition can be extended to cover the *degenerate* case where $\mathbf{K}$ is singular, but that falls outside the scope of this book. In the one-dimensional case with $\mu = \mu$ and $\mathbf{K} = \sigma^2$, this definition coincides with Definition 4.3.5.

Diagonal covariance matrices imply that the components are uncorrelated.

■ **Example 14.11** If $Z_1,\ldots,Z_d \overset{i.i.d.}{\sim} \text{Normal}(0,1)$, then $\mathbf{Z} \sim \text{Normal}(\mathbf{0}, \mathbf{I}_d)$. Then $\mathbf{Z}$ is sometimes called a *standard Gaussian vector*. ■

The standard Gaussian distribution has the attractive property of being rotationally invariant, a fact which follows from Theorem 14.3.4. Visually, if you rotate a sphere you will end up with a sphere indistinguishable from the original.

Fact 14.5.1 If $\mathbf{Z} \sim \text{Normal}(\mathbf{0}, \mathbf{I}_d)$ then $\mathbf{QZ} \sim \text{Normal}(\mathbf{0}, \mathbf{I}_d)$ for any orthogonal linear transformation $\mathbf{Q}$.

Ellipses, the SVD, and the Principal Axis Theorem (Theorem 10.4.1) all make a return:

Fact 14.5.2 For a multivariate normal distribution $f_{\mathbf{X}}$ and constant $\alpha \in (0, \max_{\mathbf{x}} f_{\mathbf{X}}(\mathbf{x}))$, the set of points for which $f_{\mathbf{X}}(\mathbf{x}) = \alpha$ is an ellipsoid in $\mathbb{R}^d$, whose principal axis directions are given by the singular vectors of $\mathbf{K}$, and whose semiaxis lengths are proportional to the singular values of $\mathbf{K}$.

■ **Example 14.12** Consider the four covariance matrices

$$\mathbf{K}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{K}_2 = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix}, \quad \mathbf{K}_3 = \begin{bmatrix} 1 & -.5 \\ -.5 & 1 \end{bmatrix}, \quad \mathbf{K}_4 = \begin{bmatrix} 1 & .8 \\ .8 & 1 \end{bmatrix}.$$

Figure 14.4 shows scatterplots of data sampled from the multivariate normal distributions corresponding to each of these covariance matrices (with mean zero), as well as the contour plots of the corresponding joint PDFs. ■

14.5.1 Correlation and Independence

Not every collection of Gaussian random variables has a multivariate normal distribution. We refer to the ones that do as being *jointly Gaussian*.

Definition 14.5.2 Two random variables X and Y are said to be *jointly Gaussian* if the vector $[X, Y]$ has a multivariate normal distribution.

■ **Example 14.13** Let $X \sim \text{Normal}(0,1)$, and let W be independent of X and take on values ± 1 with equal probability $1/2$. Let $Y = WX$. Then $Y \sim \text{Normal}(0,1)$, and X and Y are uncorrelated because $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = 0$, but any pair of values $(X(\omega), Y(\omega))$ falls on one of the two lines $y = x$ and $y = -x$. Thus X and Y do not have a joint Gaussian distribution. ■

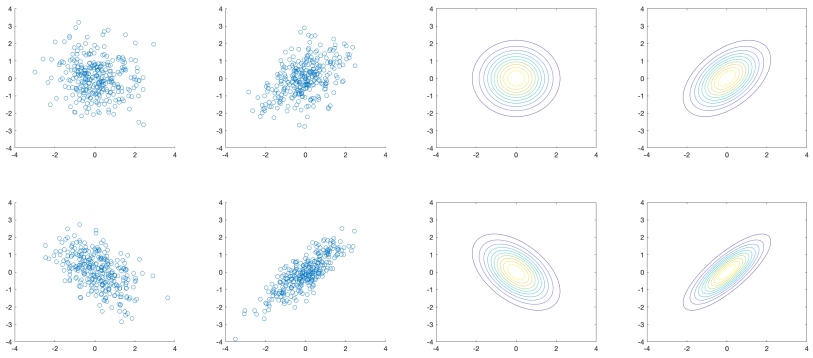


Figure 14.4: Left: Samples from Gaussian distributions with varying covariance matrices. Right: Contour plots of associated density functions.

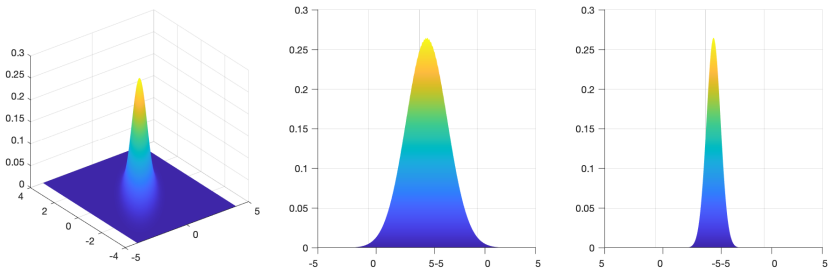


Figure 14.5: Left: Surface plot of Gaussian PDF. Middle: Distribution in first principal direction. Right: Distribution in second principal direction.

On the other hand, if a collection of variables does have a joint Gaussian distribution then it also has a number of other attractive properties.

Fact 14.5.3 A random vector $\mathbf{X} = [X_1, \ldots, X_d]$ has a multivariate normal distribution if and only if $\mathbf{a}^T\mathbf{X}$ has a Gaussian distribution for every vector $\mathbf{a}$.

The variables X and Y in Example 14.13 fail this test since $X+Y$ does not have a Gaussian distribution (it is zero with probability $1/2$, and therefore not even continuous).

Even better, uncorrelated variables are independent, which as we have seen is not true in general.

Theorem 14.5.4 Let X and Y have a joint Gaussian distribution. Then X and Y are independent if and only if $\text{Cov}(X,Y) = 0$.

The forward statement of Fact 14.5.3 can be generalized: an affine transformation of a multivariate Gaussian is still a multivariate Gaussian. Note that this is a strictly stronger claim than the one made in Theorem 14.3.4.

Theorem 14.5.5 If $\mathbf{X} \sim \text{Normal}(\mu, \mathbf{K})$ and $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}$, then $\mathbf{Y} \sim \text{Normal}(\mathbf{A}\mu + \mathbf{b}, \mathbf{A}\mathbf{K}\mathbf{A}^T)$.

14.5.2 Sampling

Theorem 14.5.5 can be used to sample from a multivariate Gaussian distribution. We start by stating an immediate consequence of that theorem.

Corollary 14.5.6 If $\mathbf{X} \sim \text{Normal}(\mu, \mathbf{K})$ and $\mathbf{K} = \mathbf{A}\mathbf{A}^T$ then $\mathbf{X} \sim \mu + \mathbf{A}\,\text{Normal}(\mathbf{0}, \mathbf{I}_n)$.

So to generate samples from a multivariate Gaussian distribution, we find such a matrix $\mathbf{A}$, generate i.i.d. standard Gaussian samples $\mathbf{Z}$ using one of the methods from Chapter 7, then return $\mathbf{X} = \mu + \mathbf{A}\mathbf{Z}$. One option for this is to use the eigenvalue decomposition or SVD of $\mathbf{K}$,

$$\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T, \quad \mathbf{A} = \mathbf{Q}\Lambda^{1/2}.$$

Another cheaper option is to use the Cholesky decomposition (Definition 8.1.3), which factors

$$\mathbf{K} = \mathbf{L}\mathbf{L}^T, \quad \mathbf{A} = \mathbf{L}$$

where $\mathbf{L}$ is lower triangular.

R In the event that you wish to generate $\mathbf{X} \in \mathbb{R}^{n\times d}$ consisting of n samples over a distribution in d dimensions, then use the transpose

$$\mathbf{X} = \mathbf{Z}\mathbf{A}^T + \mu^T$$

instead.

■ **Example 14.14** To generate a Gaussian random vector $\mathbf{X} \sim \text{Normal}(\mathbf{0}, \mathbf{K})$ with

$$\mathbf{K} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix},$$

we first generate $\mathbf{Z} \sim \text{Normal}(\mathbf{0}, \mathbf{I}_2)$, two independent standard Gaussian samples. Using the eigenvalue decomposition, we find that

$$\mathbf{A} = \mathbf{Q}\Lambda^{1/2} = \frac{1}{\sqrt{2}}\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 0.2 & 0 \\ 0 & 0.8 \end{bmatrix}^{1/2} \approx \begin{bmatrix} -0.3162 & 0.9487 \\ 0.3162 & 0.9487 \end{bmatrix}.$$

Alternately, using the Cholesky decomposition, we find that

$$\mathbf{A} = \mathbf{L} = \begin{bmatrix} 1 & 0 \\ 0.8 & 0.6 \end{bmatrix}.$$

These solutions are different but both satisfy $\mathbf{A}\mathbf{A}^T = \mathbf{K}$, which is all that is required. ■

14.5.3 Gaussian Random Fields

In many applications we might be interested in modeling a random process where points that are close to one another, either in time or space, are more strongly correlated. We might be modeling some stochastic process such as a financial market, or we might be modeling some geological feature. Either way, it is useful to define a model for which the covariance between points is greater the closer those points are to one another.

Definition 14.5.3 — Gaussian random field. A *Gaussian random field* is a (possibly infinite) collection of random variables such that any finite subset has a joint Gaussian distribution.

If each random variable $X(\mathbf{s})$ is associated with a vector $\mathbf{s} \in \mathbb{R}^m$, perhaps denoting a location in time or space, then we can define the Gaussian random field by a mean function $\mu(\mathbf{s}) = \mathbb{E}[X(\mathbf{s})]$ and a covariance function $K(\mathbf{s},\mathbf{t}) = \text{Cov}(X(\mathbf{s}),X(\mathbf{t}))$.

■ **Example 14.15** Consider the random variables $X(0), X(0.01), X(0.02), \ldots, X(10)$, each normally distributed with mean zero and variance one, having covariance function

$$K(s,t) = \exp(-\gamma(s-t)^2)$$

for some parameter $\gamma > 0$. When $\gamma = 1$ for example, points at a distance of 1 will have covariance $\exp(-1) \approx 0.37$ and points at a distance of 2 will have covariance $\exp(-4) \approx 0.018$. When $\gamma = 10$, the covariance decays more quickly. Points at a distance of 1 will have covariance $\exp(-1) \approx 4.5 \cdot 10^{-5}$, or nearly zero. Figure 14.6 shows one sample from the random field using each of these two parameters. Even though 1001 random samples have been generated, the resulting plots are fairly smooth because of the strong correlation between nearby points.

The method outlined in the previous section, factoring $\mathbf{K} = \mathbf{A}\mathbf{A}^T$ and computing $\mathbf{X} = \mathbf{A}\mathbf{Z}$ for i.i.d. Gaussian samples $\mathbf{Z}$, requires us to produce 1001 random numbers—the same as the number of outputs. However, the eigenvalues of $\mathbf{K}$ decay rapidly, and so the matrix has a good low-rank approximation. By using such an approximation $\mathbf{K} \approx \mathbf{A}_r\mathbf{A}_r^T$ and taking $\mathbf{X} \approx \mathbf{A}_r\mathbf{Z}_r$, we can get away with computing only r samples (and a much cheaper matrix factorization) instead. Figure 14.7 shows that for a modest $r = 38 \ll 1001$, we capture over 99 percent of the variance of the original model. ■

In the case of the specific covariance function in the example above, a similar effect could be produced by generating independent Gaussian variables and applying blurring, where a greater degree of blurring would cause variables remain correlated at greater distances.

Figure 14.8 shows a similar example of random sampling from a Gaussian random field in two dimensions.

14.6 Matlab Commands

Covariance

- `cov(X)`: Returns the covariance matrix of `X`, where each row is an observation and each column is a varible. If `X` is a vector, returns the variance.
- `cov(X,Y)`: If `X` and `Y` are the same shape, returns the 2×2 covariance matrix `cov([X(:),Y(:)])`.
- `corrcoef(X)`: Returns the correlation coefficients of `X`.

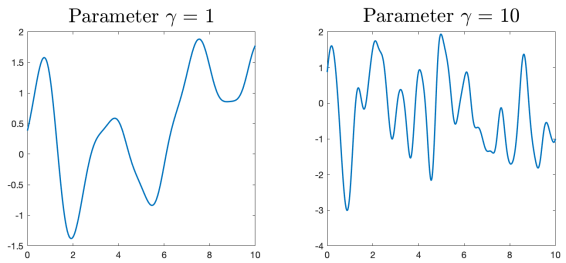


Figure 14.6: A smaller parameter γ results in variables remaining highly correlated at longer distances.

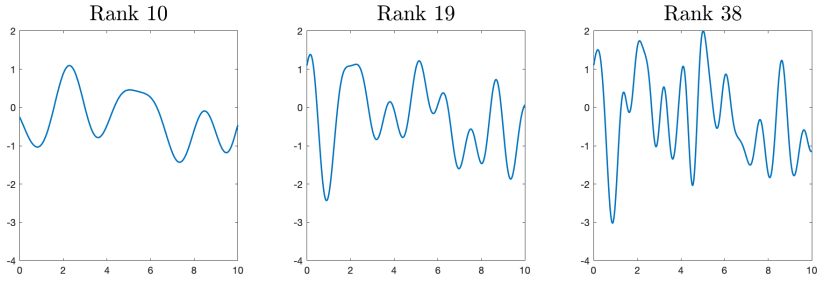


Figure 14.7: Parameter $\gamma = 10$. When the covariance matrix has a low rank approximation, the variables can be approximately simulated at a lower cost. Plot capture 51 percent, 81 percent, and 99 percent of the variance of the original model, respectively.

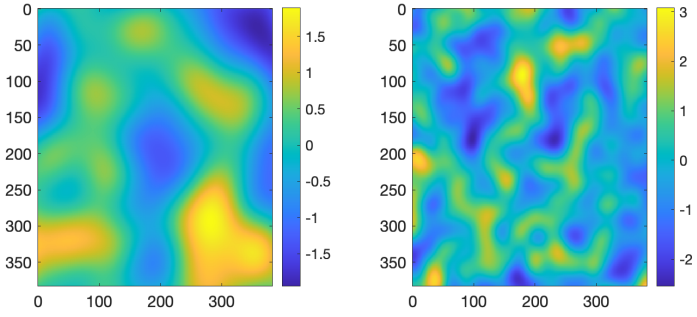


Figure 14.8: Sample from a Gaussian random field in two dimensions. In the left plot, variables remain correlated at greater distances.

- `corrcoef(X,Y)`: If `X` and `Y` are the same shape, returns the 2×2 matrix of correlation coefficients `corrcoef([X(:),Y(:)])`.

Multivariate Normal Distribution

- `mvnrnd(mu,Sigma)`: Generates random vectors from the multivariate normal distribution.
- `mvnpdf(x,mu,Sigma)`: Multivariate normal PDF.
- `mvncdf(x,mu,Sigma)`: Multivariate normal CDF.

15. Clustering

Time for some machine learning! What is that, you ask? We'll define it roughly as building mathematical models on sample data for the purpose of making predictions or decisions. We'll take a look at two broad classes of learning algorithms in particular:

- *Supervised learning*, in which we train the model on labeled data (or more generally, data where the desired output is known).
  - If we have a set of measurements on irises whose species we have already identified, can we use this data to determine the species of the next flower we encounter?
  - Netflix has information on how millions of users have rated thousands of movies. Can they give good recommendations for a given user?
  - Given a large corpus of English books and their French translations (or vice versa), can the computer be trained to translate a new sentence from one language to the other?
  - Given a collection of images of handwritten letters and digits and their correct labels, can a mail-sorting machine sort the mail accurately?
  - Given a large amount of training data, can a self-driving car tell the difference between a red light and a green light?
- *Unsupervised learning*, in which the input data is not labeled. We might nonetheless be interested in detecting patterns in the data for the purpose of making future inferences.
  - If we encounter an unknown writing system, can we determine whether the system is logographic, or if it uses an alphabet, abjad, or syllabary?
  - If we have a set of iris measurements but do not know the species, can we guess at how many distinct species there might be?
  - During the London bombing campaign in World War II, the British government kept an accurate census of where in the city bombs fell [2]. Do the data suggest that the German bombers could pinpoint targets accurately, or could the distribution be explained by chance alone?

A class in between these two is *semi-supervised learning*, where some of the data is labeled but some is not.

One common motivation for developing machine learning algorithms is to be able to carry out tasks at a scale where using humans would be prohibitively expensive. We may not be able to train a computer to give personalized movie recommendations as well as an experienced human could (at least, not at the present time), but it would be impractical for a streaming service to hire humans to give individual recommendations for each of their millions of subscribers. One notable success in this area is in identifying handwritten digits: on the MNIST benchmark, state-of-the art algorithms have over a 99.8 percent accuracy rate. This book won't get into the more technical details of these algorithms, but we will at least aim to set the foundation for them, starting with clustering.

15.1 K-Means Clustering

Many different clustering algorithms exist and the proper algorithm to use may depend strongly on the situation. Our focus will be on one known has *k-means clustering*, which represents clusters by a single central point and uses the idea that in a "good" clustering, each data point should be as close to that central point as possible.

Put more formally: for a set

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

of vectors in $\mathbb{R}^d$, we fix a target number of clusters k and seek a partitioning of S

$$S = S_1 \sqcup S_2 \sqcup \cdots \sqcup S_k,$$

where the symbol $\sqcup$ denotes a disjoint union: $S_i \cap S_j = \emptyset$ when $i \neq j$. For simplicity, we represent the partitioning as a set of sets

$$\mathscr{S} = \{S_1, S_2, \ldots, S_k\}.$$

To the disjoint sets we assign central points

$$M = \{\mu_1, \ldots, \mu_k\}$$

so that for each $1 \leq j \leq k$ the points in S_j are as close to the central point as possible.

How should we measure closeness? By the squared Euclidean distance, of course! Now "of course" doesn't mean that this measure of distance necessarily gives us the best results, but it does make the problem quite a bit simpler mathematically and aligns nicely with the goal of minimizing the variance within each cluster. In particular, the optimal "central points" will simply be the arithmetic means of the data points within each set, courtesy of the following extension of Fact 12.4.1:

Fact 15.1.1 For a set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of vectors in $\mathbb{R}^d$, the arithmetic mean μ minimizes the sum-of-squares distance $\sum_{i=1}^n \|\mathbf{x}_i - \mu\|_2^2$.

With this fact in hand, we can write the k-means clustering problem as

$$\min_{\mathscr{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|_2^2, \qquad \mu_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x}. \tag{KMEANS}$$

15.1.1 Complexity

There is one small wrinkle: not only is KMEANS not a linear least squares problem, it's combinatorial in nature since we have to decide how to partition the set S.

Fact 15.1.2 For fixed number of clusters k and dimension d, problem (KMEANS) can be solved in $\mathcal{O}(n^{dk+1})$ time.

This is not a particularly helpful guarantee. Even if we have just two clusters and the data is two-dimensional, Fact 15.1.2 promises that we can find the exact solution in time $\mathcal{O}(n^5)$, which is cause for concern as the number of points to be clustered grows. By comparison, the most expensive linear algebra problems we considered could be solved in $\mathcal{O}(n^3)$ time, and problem sizes n of more than a few thousand could be time-consuming for a laptop. Although the big-Oh notation does not tell us the constants involved in these costs, finding the exact solution to KMEANS when n is in the millions seems out of the question.

The worst-case bounds look even worse when the number of dimensions or the number of clusters increase.

Fact 15.1.3 Even for $k = 2$ clusters, KMEANS is NP-hard with respect to d.

Fact 15.1.4 Even for $d = 2$ dimensions, KMEANS is NP-hard with respect to k.

We won't go into the technical meaning of "NP-hard", but for our purposes the implication is that the cost is generally thought to grow exponentially with respect to d and k. It's not known for certain that no polynomial-time algorithm exists, but if you find one you will have solved one of the most famous open problems in mathematics [11]. For this reason, practical algorithms for solving KMEANS aim to provide only an *approximate* solution, rather than an exact one.

15.1.2 Naive k-Means

The standard algorithm for solving KMEANS, known as *naive k-means* or *Lloyd's Algorithm*, was proposed by Stuart Lloyd in 1957. One way to derive this algorithm is to rewrite the k-means problem as

$$\min_{\mathcal{S}} \min_{M} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|_2^2,$$

and observe that while it may be difficult to optimize over $\mathcal{S} = \{S_i\}_{i=1}^k$ and $M = \{\mu_i\}_{i=1}^k$ *simultaneously*, it is simple to optimize over either one of the two variables *while the other is fixed*. Namely,

- For a fixed set of points M, the optimal partition assigns each point x_j $(1 \leq j \leq n)$ to the closest μ_i;
- For a fixed partition $\mathcal{S}$, the optimal M is found by setting each μ_i to be the arithmetic mean of the elements in S_i, as stated in Fact 15.1.1.

Naive k-means then pursues a strategy of *alternating minimization*: it solves for the optimal means M, then finds the optimal partition $\mathcal{S}$ for those means, then finds the means M for the new partition, and so on and so forth. One can continue iterating until some user-defined maximum number of iterations is reached, or until the sum-of-squares distance no longer decreases by a substantial amount. In the latter case, it would also be up to the user to define what amount of improvement counts as "substantial". Figure 15.1 shows the alternating minimization process on the Iris dataset, where the "A" step refers to recomputing the clusters and the "B" step refers to recomputing the means. The starting values for the means were chosen randomly.

The naive k-means algorithm has several attractive features. It relatively simple to implement, it costs only $\mathcal{O}(ndk)$ per iteration (computing the 2-norm $\|\mathbf{x}_j - \mu_i\|_2$ in d dimensions for $1 \leq j \leq n$ and $1 \leq i \leq k$), and it is guaranteed to halt in finite time.

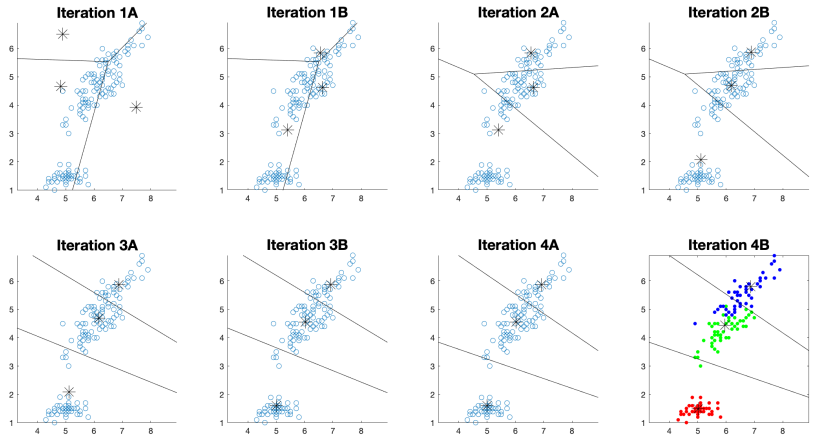


Figure 15.1: Naive k-means algorithm run on Iris data in 2 dimensions. Sepal length is given on the x-axis and petal length on the y-axis. In the final plot, true species are shown in color.

Theorem 15.1.5 — Naive k-means. The naive k-means algorithm halts after a finite number of iterations.

Proof. The sum-of-squares distance descreases monotonically each time either the clusters or the means are recomputed. The choice of clusters must therefore be different at each iteration, and there are only a finite number of possible partitions of the data set. ■

On the other hand, the "finite time" guarantee is not all that great. In the worst-case scenario as the number of samples n increases, there exists a constant $c > 1$ so that naive k-means requires at least $c^{\sqrt{n}}$ iterations to converge. In the limit, this expression grows faster than any polynomial n^c, so it is not generally worth it to run k-means until it has converged exactly.

15.1.3 Limitations

The naive algorithm for k-means clustering has a few significant limitations. First, although the algorithm is guaranteed to converge, it does not necessarily converge to the optimal solution. It can't quite be said that the problem is non-convex in the sense of Definition 12.1.1 because the choice of partitioning $\mathcal{S}$ is a discrete problem rather than a continuous one, but the effect is similar: unless the starting guess is chosen well, the algorithm might get caught in a local minimum that is significantly worse than the optimal solution. Figure 15.2 shows an example of this phenomenon, where one cluster is well-separated from the other two. For this reason, variants such as *k-means++* have been developed which choose the starting points more carefully. While they do not necessarily converge to the optimal solution either, these more sophisticated methods offer better guarantees on how far from optimal the eventual clustering can be.

Ideally, the clusters have roughly spherical distributions with similar populations and variance. If any of these conditions fail, k-means clustering may give less than ideal results. Figure 15.3 illustrates a few such scenarios. The simplest of these problems to correct for is when the distributions are oblong rather than spherical, since we can always start by rescaling each dimension of the data to have unit variance. When the populations or variances are different, a Gaussian mixture model (discussed later in this chapter) would be more appropriate. When the data are non-Gaussian, an entirely different sort of clustering algorithm is needed.

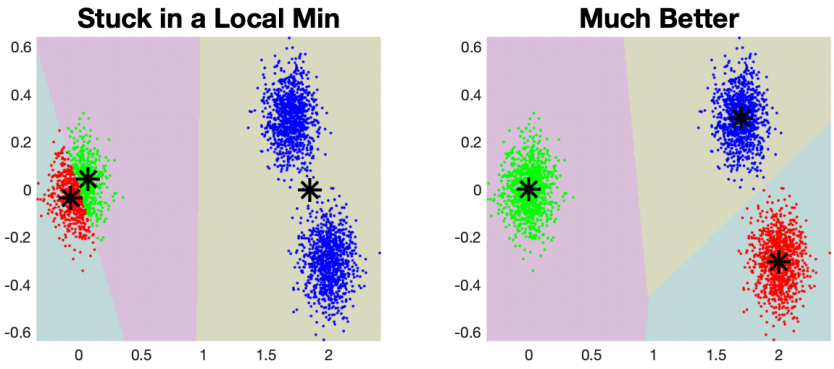


Figure 15.2: With a poor choice of starting means, the naive k-means algorithm can converge to a local minimum that is far from optimal.

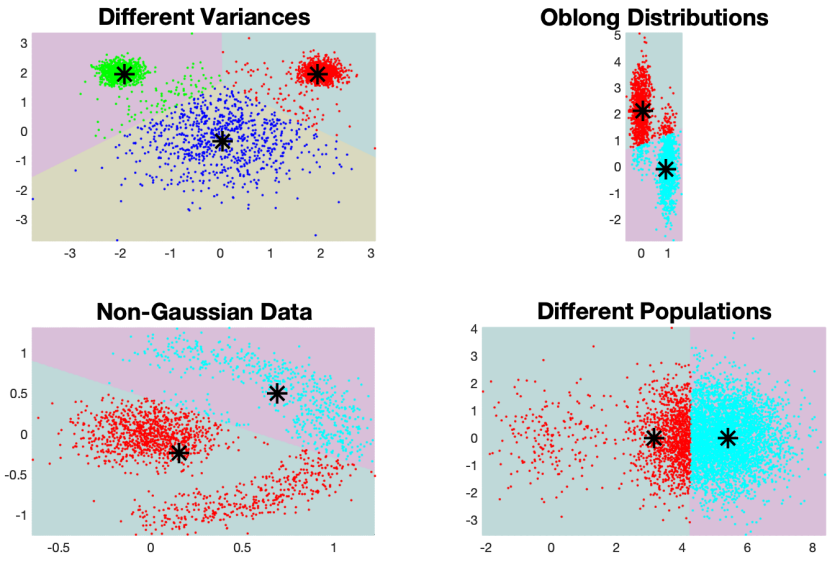


Figure 15.3: A variety of scenarios in which k-means might give poor results.

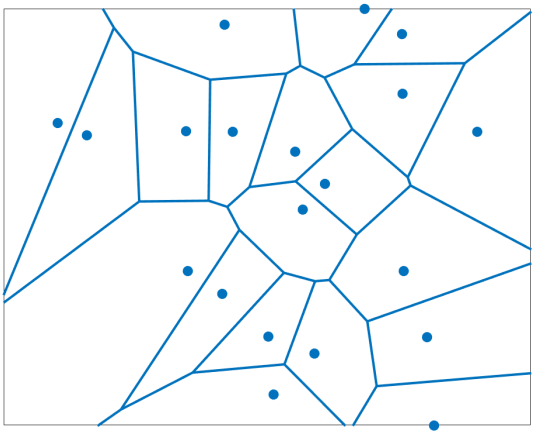


Figure 15.4: Voronoi diagram on random data. Decision boundaries are straight lines.

15.1.4 Voronoi Diagrams

It is worth looking into the failure of k-means on non-Gaussian data a little more closely. We start by observing that any choice of means $\{\mu_1, \ldots, \mu_k\}$ splits the domain into what is known as a *Voronoi Diagram*.

Definition 15.1.1 — Voronoi Diagram. A *Voronoi Diagram* determined by a set of points $\{\mu_1, \ldots, \mu_k\}$ in $\mathbb{R}^d$ is collection of k *Voronoi cells* $\mathbb{R}^d = R_1 \cup R_2 \cup \cdots \cup R_k$ such that each R_i is the set of all points at least as close to μ_i as to any other point μ_j. If the distance function used is the Euclidean distance,

$$R_i = \{\mathbf{x} \in \mathbb{R}^d \ : \ \|\mathbf{x} - \mu_i\|_2 \leq \|\mathbf{x} - \mu_j\|_2 \text{ for all } j \neq i\}.$$

Figure 15.4 shows an example for 20 randomly scattered points. Each point in $\mathbb{R}^d$ would then be assigned to the cluster associated with a given mean μ_i if and only if the point lies within the associated Voronoi cell R_i.

Not all points in the space can be definitively assigned to a particular cluster.

Definition 15.1.2 A *decision boundary* is the region in $\mathbb{R}^d$ for which the classification of a point is ambiguous.

In the case of k-means clustering, the decision boundary consists of all points that are equidistant from two or more closest means μ_i. For the Euclidean norm in particular, the decision boundaries are necessarily linear.

Theorem 15.1.6 For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, the set of points equidistant from $\mathbf{a}$ and $\mathbf{b}$ is given by the hyperplane

$$H = \left\{ \frac{\mathbf{a}+\mathbf{b}}{2} + \mathbf{z} \ : \ (\mathbf{a}-\mathbf{b})^T \mathbf{z} = 0 \right\} = \frac{\mathbf{a}+\mathbf{b}}{2} + \mathcal{N}\left((\mathbf{a}-\mathbf{b})^T\right). \tag{15.1}$$

This hyperplane is the pendicular bisector of the line segment from $\mathbf{a}$ to $\mathbf{b}$.

Proof. Equality of distances $\|\mathbf{x}-\mathbf{a}\|_2^2 = \|\mathbf{x}-\mathbf{b}\|_2^2$ holds if and only if

$$(\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a}) = (\mathbf{x}-\mathbf{b})^T(\mathbf{x}-\mathbf{b}).$$

Figure 15.5: Some digits can be written in more than one style.

Distributing the terms and rearranging yields the linear system

$$2(\mathbf{a}-\mathbf{b})^T\mathbf{x} = (\mathbf{a}-\mathbf{b})^T(\mathbf{a}+\mathbf{b}),$$

which has exactly the set of solutions given in 15.1. ■

It follows that the boundary of each Voronoi cell is defined by a collection of hyperplanes, or more formally, that every Voronoi cell is *the intersection of the half-spaces* defined by these hyperplanes. In other words:

Fact 15.1.7 Every Voronoi cell R_i (using the Euclidean distance) is a convex polyhedron.

Here, in a use related to but distinct from Definition 12.1.1, "convex" means that if two points are in a set then so is the entire line segment connecting them. In two dimensions, a polygon is convex if and only if all of its internal angles are less than 180 degrees.

Definition 15.1.3 A set $S \subseteq \mathbb{R}^d$ is *convex* if for all $\mathbf{x}, \mathbf{y} \in S$ and all $t \in [0,1]$, the point $t\mathbf{x}+(1-t)\mathbf{y}$ is also in S.

In the case of Voronoi cells, the justification for the convexity claim is that the set of all points on one side of a hyperplane (i.e., the half-space) is convex, and the intersection of multiple convex sets is also convex.

In any case, the kicker is that since k-means clustering necessarily splits the domain into convex cells, it cannot possibly give a satsisfactory clustering when the data cannot be separated by a linear boundary, as is the case for the non-Gaussian data in Figure 15.3.

15.1.5 Number of Clusters

One last practical problem we encounter when using k-means clustering faces is that of deciding how many clusters to use. For example, it is not obvious from looking at the unlabeled Iris dataset whether there are two different species of flower or three. Even in cases where we know the number of underlying classes, the best number of clusters to set might be clear. As an example, take the MNIST dataset (Section 11.3.4): even though we know that there are exactly ten digits, some digits can be written in more than one way (see Figure 15.5). Does it make more sense to treat these different instances as a single cluster or to attempt to handle them separately?

A rough heuristic is to try running the algorithm for multiple different values of k and to examine the value of the objective function in KMEANS as a function of k. With any luck, there will be an obvious kink in the graph representing a point where increasing the number of clusters further suddenly faces decreasing marginal returns.

■ **Example 15.1** Figure 15.6 shows results for the Iris and MNIST datasets, where the sum-of-squares of all distances from points to their cluster means are plotted against the number of clusters

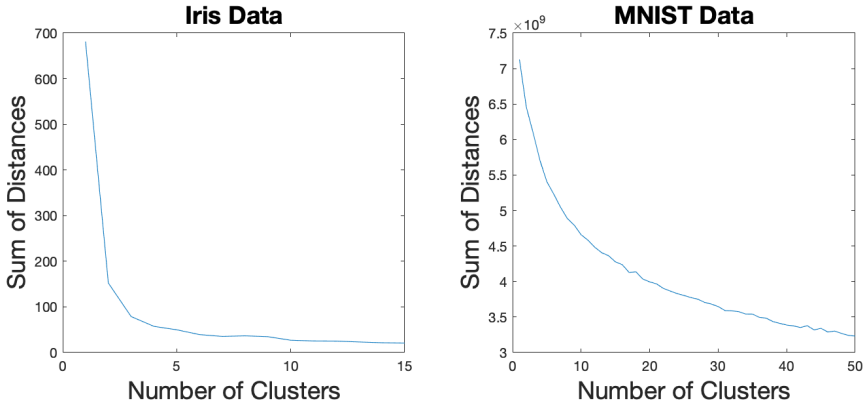


Figure 15.6: Graphs showing the sums-of-squares of all distances from points to their cluster means. Right: MNIST dataset reduced to 2 dimensions.

(the MNIST dataset was first reduced to 2 dimensions using PCA). The graph for the Iris data shows an obvious kink and suggests that it would probably be appropriate to use two or three clusters. The graph for the MNIST data is less clear-cut. ∎

15.2 Curse of Dimensionality

It is worth considering that we might have gotten better results for the MNIST set in Example 15.1 if we had not used PCA to reduce the data set down to two dimensions first. Reducing the dimensionality might be good for visualizing the data, but why would we toss out data that could be useful for distinguishing points from one another? It's clear from Figure 15.6 that two dimensions is not enough to separate all of the digits cleanly, so why not just use more?

Well, the reduction *was* for visualization purposes and we almost certainly should be using more dimensions. But there's a limit to how much the extra dimensions of information can help us, due to the *curse of dimensionality*. This term doesn't refer to any one specific theorem but instead is a catch-all for a variety of ways that high-dimensional spaces behave strangely compared to our expecations. Roughly, much of our intution for how distances and volumes work will fail as the number of dimensions increases.

For example, in two dimensions the unit circle has an area of $\pi \approx 3.14$ and in three dimensions the unit sphere has a volume of $(4/3)\pi \approx 4.19$. As the number of dimensions increases, the volume of the unit ball in d dimensions (Definition 3.4.3) increases... but only up to $d = 5$ where it hits a maximum of $(8/15)\pi^2 \approx 5.26$. After that, the volume start to drop.

Theorem 15.2.1 — Hypersphere. The volume of the unit sphere in d dimensions is given by

$$V(d) = \frac{2\pi^{d/2}}{d\Gamma(d/2)},$$

where Γ is the *gamma function*. As the number of dimensions increases,

$$\lim_{d\to\infty} V(d) = 0.$$

Things look even stranger if we compare these hyperspheres to their enclosing hypercubes, which have a volume of 2^d. As the number of dimensions increases, the ball takes up an tinier and tinier portion of the cube!

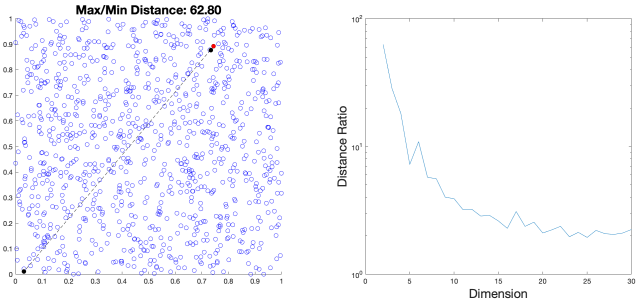


Figure 15.7: Curse of dimensionality. When points are randomly scattered in high dimensions, the closest and farthest points are approximately the same distance from a given reference point. Left: two dimensions. Reference point shown in red; closest and farthest points shown in black.

The specific phenomenon that we are interested in here is related to a consequence of the Central Limit Theorem. As we sample points randomly from distributions of higher and higher dimensions, the Euclidean distance become less helpful in determining the relative closeness between pairs of points.

Theorem 15.2.2 — Curse of Dimensionality. Fix a probability distribution, and number of samples n. For a given number of dimensions d, generate random vectors $\mathbf{X} \in \mathbb{R}^d$ with i.i.d. components drawn from the chosen distribution. For random points $\mathbf{X}_0, \mathbf{X}_1, \ldots \mathbf{X}_n$, use $\mathbf{X}_0$ as a reference point and define

$$\text{dist}_{\max} = \max_{1 \le i \le n} \|\mathbf{X}_0 - \mathbf{X}_i\|_2, \qquad \text{dist}_{\min} = \min_{1 \le i \le n} \|\mathbf{X}_0 - \mathbf{X}_i\|_2.$$

Then $\text{dist}_{\max}$ and $\text{dist}_{\min}$ are themselves random variables whose distributions depend on the number of dimensions d. Then

$$\lim_{d \to \infty} \mathbb{E}\left[\frac{\text{dist}_{\max}(d)}{\text{dist}_{\min}(d)}\right] = 1.$$

As the number of dimensions increases, almost all points will be roughly equally far from the reference point.

Figure 15.7 shows an illustration for $N = 1000$ points randomly scattered in the unit hypercube $[0,1]^d$ for dimensions $2 \le d \le 30$. In two dimensions the farthest point from the reference point is over sixty times farther from the closest point. For $d = 30$, it is only slightly more than twice as far away. In order to make the Euclidean norm a more effective measure of distance and in order to reduce the influence of noise in the data, it is therefore common to use PCA or other related methods for reducing the dimensionality of the data set before carrying out further analysis.

Another way of thinking about high-dimensional spaces is to consider how many points it would take to properly "fill out" the space. As a starting point, we make the following observation.

Fact 15.2.3 Dividing the unit hypercube $[0,1]^d$ in half along each coordinate axis will split the cube into 2^d regions.

So a line has two halves, a square has four quadrants, a cube has eight octants, and so on. If we do not at least have $n \ge 2^d$ data points, then there will be regions of the hypercube containing only one point, or none! So at the very least it's a good idea to use a data set large enough that $n \gg 2^d$.

This observation calls back to our motivation for using Monte Carlo integration back in Chapter 6: other integration strategies such as the Trapezoid method may converge faster *in the limit*, but are computationally intractable in high dimensions.

15.3 Single-Linkage Clustering

A rather different type of clustering is known as *hierarchical* clustering, which builds a hierarchy of clusters by producing from the data a *dendrogram* (similar to a phylogenetic tree in biology). We'll take a specific look at *single-linkage* clustering, which uses distances between individual pairs of points to build the tree from the bottom up.

The method begins by putting each data point in its own cluster of size one. It then finds the closest two clusters, and merges them into a single cluster. It continues merging clusters in this manner until all data have been grouped into a single cluster. For single-linkage clustering, the distance between two clusters is said to be equal to the smallest distance between any pair of points in the clusters (one point from each cluster).

Definition 15.3.1 The single-linkage Euclidean distance between two clusters A and B is given by

$$d(A,B) = \min\{\|\mathbf{x}-\mathbf{y}\|_2 \ : \ \mathbf{x} \in A, \mathbf{y} \in B\}.$$

Other variants exist. *Complete linkage clustering* considers the maximum distance between the points in the clusters instead of the minimum, and we could also consider a weighted average of all pairwise distances between points, or a metric other than the Euclidean distance.

15.3.1 Advantages and Disadvantages

One potential advantage of hierarchical clustering over k-means in general is that it provides the entire dendrogram rather than a single clustering. Clusters are joined in order of their distance, where links with greater y-values in the dendrogram represent greater distances between the clusters being joined (see Figure 15.8). One could potentially use this information to determine how many clusters should be used to group the data. Another advantage of single-linkage clustering is its ability to handle data sets that cannot be partitioned by linear boundaries.

One disadvantage of single-linkage clustering is that it has does not handle outliers well, and has a tendency to assign them to their own private clusters (see Figure 15.9). One could perhaps sell this as a feature instead of a bug and say that it is good at outlier detection, but at the very least some additional processing steps would be needed. More seriously, because it only examines one link at a time, single-linkage clustering may follow a chain of short distances to connect two very different clusters together (this is sometimes called the *single-link effect*, or "chaining phenomenon").

15.3.2 Cost

Assuming that all pairwise distances between the points have been precomputed (this would cost $\mathcal{O}(n^2 d)$ in d dimensions), single-linkage clustering requires $\mathcal{O}(n^2)$ memory to store all pairwise distances. The naive algorithm for merging the clusters would cost $\mathcal{O}(n^3)$, but optimally efficient algorithms requiring $\mathcal{O}(n^2)$ time.

15.4 Matlab Commands

Clustering

- `kmeans(X,k)`: Partitions the data matrix `X` into `k` clusters using k-means. Various options control the distance function used, the maximum number of iterations, and the method for choosing starting values.

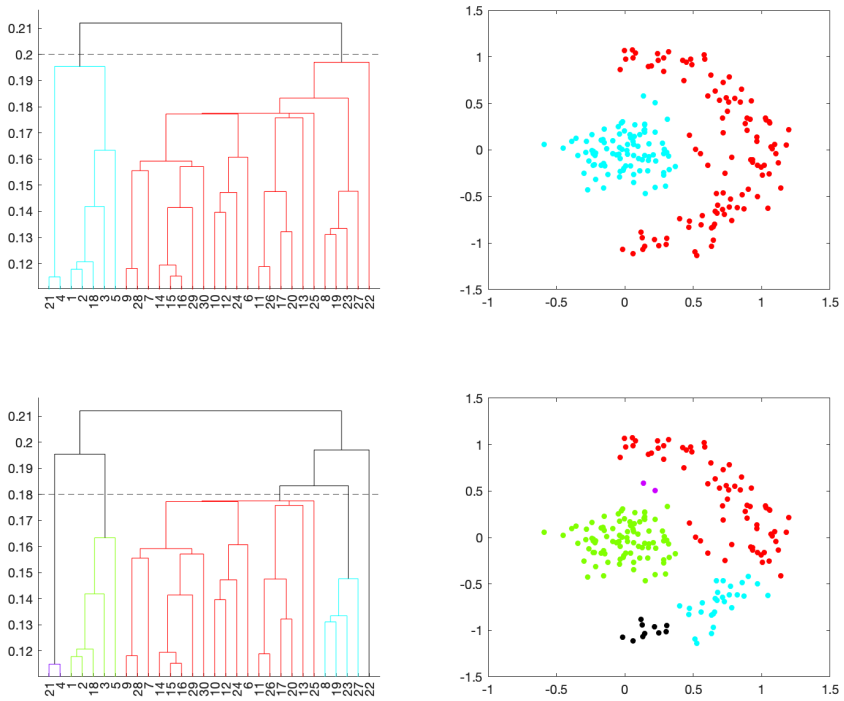


Figure 15.8: Single-linkage clustering on data from non-Gaussian distributions.

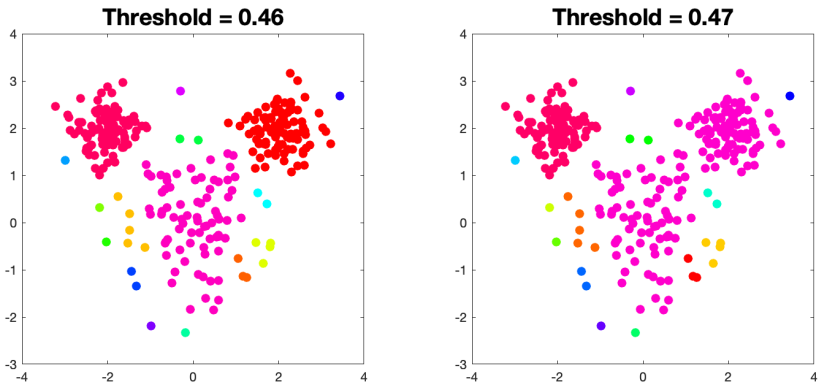


Figure 15.9: Many clusters have only a single element. Increasing the distance threshold further would connect the primary clusters to each other before it connected the outliers.

- `pdist(X)`: Returns the pairwise distances between the observations in `X`. Various different distance functions are available. Answer is returned as a single row vector, which can be converted to and from matrix format using `squareform`.
- `pdist2(X,Y)`: Returns the pairwise distances between two sets of observations. Various different distance functions are available.
- `voronoi(X,Y)`: Plots the Voronoi diagram for the points `X` and `Y`.
- `normalize`: Normalizes data according to a variety of methods. Default is to set columns to have mean zero and standard deviation 1.
- `Z = linkage(X)`: Creates a hierarchical cluster tree from the data in `X`.
- `cluster(Z,'Cutoff',c`: Clusters the data from a hierarchical cluster tree `Z` according to cutoff threshold `c`.
- `dendrogram(Z)`: Plots a dendrogram from the hierarchical cluster tree `Z`.

16. Expectation Maximization

One limitation of the clustering methods discussed in the previous chapter is that they do not quantify any uncertainty in the cluster assignments. A given point is assigned to one cluster or another, and the algorithm says nothing about its confidence in the assignment. We will consider a different type of clustering algorithm that assigns a probability distribution to the data being clustered, but first we need a few more tools from probability theory.

16.1 Conditional Probability

We need to be able to describe how our understanding of a situation might change when we learn new information.

Definition 16.1.1 — Conditional Probability of Events. For events E and F, the *conditional probability* of E given F is

$$\mathbb{P}(E|F) = \frac{\mathbb{P}(E \cap F)}{\mathbb{P}(F)}.$$

■ **Example 16.1** Roll a die. Let E be the event that we roll a 6 and let F be the event that we roll an even number. Then

$$\mathbb{P}(E|F) = \frac{1/6}{1/2} = 1/3.$$

If the die has been rolled and you learn that the number rolled was even, then there is a 1/3 chance that a 6 was rolled. ■

■ **Example 16.2** Let E be the event that it snowing in Raleigh and let F be the event that it is above sixty degrees in Raleigh. Then $\mathbb{P}(E|F) = 0$. If you learn that the temperature is above sixty, then it is definitely not snowing. ■

Definition 16.1.2 — Conditional Probability Distributions. For discrete random variables X and Y, the conditional PMF of Y given X is defined as

$$p_{Y|X}(y|x) = \mathbb{P}(Y = y|X = x) = \frac{\mathbb{P}(X = x \text{ and } Y = y)}{\mathbb{P}(X = x)}.$$

For continuous random variables with a joint probability density function, the conditional PDF is

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x,y)}{f_X(x)},$$

where $f_{X,Y}$ is the joint PDF and f_X is the PDF of X alone.

■ **Example 16.3** Roll two dice X and Y and let $S = X + Y$. Then

$$p_{S|Y}(5|3) = \mathbb{P}(S = 5|Y = 3) = \frac{\mathbb{P}(S = 5 \text{ and } Y = 3)}{\mathbb{P}(Y = 3)} = \frac{1/36}{1/6} = 1/6.$$

If Y rolled a 3, there is a 1 in 6 chance that $X + Y = 5$. On the other hand,

$$p_{S|Y}(10|3) = \mathbb{P}(S = 10|Y = 3) = \frac{\mathbb{P}(S = 10 \text{ and } Y = 3)}{\mathbb{P}(Y = 3)} = \frac{0}{1/6} = 0.$$

If Y rolled a 3, there is zero chance that $X + Y = 10$. ■

Fact 16.1.1 If X and Y are independent random variables, then the conditional distribution of Y given X is the same as the distribution of Y. Additional information about X does not tell us anything about Y.

■ **Example 16.4** Let X and Y be independent rolls of the dice. Then for any $m, n \in \{1, \ldots, 6\}$, we have

$$\mathbb{P}(X = m|Y = n) = \frac{\mathbb{P}(X = m \text{ and } Y = n)}{\mathbb{P}(Y = n)} = \frac{1/36}{1/6} = 1/6 = \mathbb{P}(X = m).$$

The variables X and Y are independent, so the conditional distribution of X given Y is the same as the original distribution of X. ■

16.1.1 Total Probability and Bayes' Theorem

In many situations, we can easily determine conditional probabilities but need to use this knowledge to determine other probabilities. Here, the *law of total probability* can help.

Theorem 16.1.2 — Law of Total Probability. For any event E and finite or countable collection of mutually exclusive events $\{F_i : i = 1, 2, \ldots\}$ which partition the sample space, $\bigcup_i F_i = \Omega$, it holds that

$$\mathbb{P}(E) = \sum_i \mathbb{P}(E \cap F_i) = \sum_i \mathbb{P}(E|F_i)\mathbb{P}(F_i).$$

■ **Example 16.5** You have a bag containing one six-sided die and two twenty-sided dice. Assuming you pick a die out uniformly at random and that the dice are fair, what is the probability that you roll a 4?

Let N represent the number of sides on the die and let D represent the outcome of the roll. Then

$$\mathbb{P}(N = 6) = 1/3, \quad \mathbb{P}(N = 20) = 2/3, \quad \mathbb{P}(D = 4|N = 6) = 1/6, \quad \mathbb{P}(D = 4|N = 20) = 1/20.$$

Therefore, by the Law of Total Probability,

$$\mathbb{P}(D=4) = (1/3)(1/6) + (2/3)(1/20) = 1/18 + 1/30 = 4/45.$$

■

No introduction to conditional probabilities would be complete without mentioning Bayes' theorem, which allows use to make inferences based on observed data and knowledge of prior probabilities.

Theorem 16.1.3 — Bayes' Theorem. For two events E and F,

$$\mathbb{P}(E|F) = \frac{\mathbb{P}(F|E)\mathbb{P}(E)}{\mathbb{P}(F)}.$$

Here $\mathbb{P}(E)$ is often called the *prior probability*, and $\mathbb{P}(E|F)$ the *posterior probability* of E given F.

■ **Example 16.6** With the same dice as in Example 16.5, you roll a random die and are told that you rolled a 4. What is the probability that you grabbed the six-sided die?

Using Bayes' Theorem and the answer from the previous example, we find that

$$\mathbb{P}(N=6|D=4) = \frac{\mathbb{P}(D=4|N=6)\mathbb{P}(N=6)}{\mathbb{P}(D=4)} = \frac{1/18}{1/18+1/30} = 5/8.$$

Despite the fact that there was a greater chance of grabbing a 20-sided die, the roll of 4 makes it more likely on the balance that ■

One computational shortcut for this problem involves writing the prior and posterior probabilities in terms of *odds*.

Definition 16.1.3 For an event A and its complement A^c, the *odds in favor* of A can be represented by an expression $p:q$, meaning that the probability of A is $p/(p+q)$ and the probability of A^c is $q/(p+q)$. The *odds against* A would be represented by $q:p$ instead.

For example, a horse with $40:1$ odds against in a race is estimated to have a $1/41$ chance of victory. In the case of the example with the dice, we can represent the *prior odds* for $N=6$ against $N=20$ as $\frac{1}{3}:\frac{2}{3}$ and the *posterior odds* after rolling a 4 are

$$\left(\frac{1}{3}\right)\left(\frac{1}{6}\right) : \left(\frac{2}{3}\right)\left(\frac{1}{20}\right) = \frac{1}{18} : \frac{1}{30}.$$

Multiplying both sides of a set of odds by a constant does not change the underlying probabilities. We can therefore quickly rescale by multiplying both sides by $30 \cdot 18$ to get

$$\frac{1}{18} : \frac{1}{30} = 30 : 18 = 5 : 3,$$

which represents a $5/(5+3) = 5/8$ probability that we picked the six-sided die.

It's something of a cliché to mention the Monty Hall problem at this point, so we'll instead present a similarly counterintuitive variant.

■ **Example 16.7** You have a bag of coins. One third of them are regular coins, one third are trick double-headed coins, and the remaining third are trick double-tailed coins. You pull out a coin at random, flip it, and get heads. What is the probability that the other side of the coin will show tails?

Intution might suggest that since the bag contains an equal number of regular and double-headed coins, the posterior odds are equally likely. This is incorrect because it does not account for the fact

that the double-headed coins are guaranteed to land on heads while a regular coin only has a half chance of doing so. In order, the prior odds of drawing a HT/HH/TT coin are $(1/3) : (1/3) : (1/3)$, and the conditional probabilities of getting heads are $1/2$, 1, and 0. The posterior probabilities of each coin are, by Bayes' Theorem,

$$(1/3)(1/2) : (1/3)(1) : (1/3)(0) = 1/6 : 1/3 : 0 = 1 : 2 : 0,$$

so as it turns there is only a $1/3$ chance that the coin is a standard one, and a $2/3$ chance that it is double-headed. ∎

16.1.2 Common Events Occur Commonly

A common aphorism in the medical world was coined by Theodore Woodward as advice to his interns,

> When you hear hoofbeats behind you, don't expect to see a zebra.

as caution against the rookie tendency to overdiagnose rare and interesting diseases that they had recently learned about. Put another way, when the prior odds are overwhelmingly in favor of one outcome, that outcome will often still be favored in the posterior odds. Or: a rare manifestation of a common disease is often more likely than a common manifestation of a rare disease. Related is the aphorism popularized Carl Sagan,

> Extraordinary claims require extraordinary evidence.

∎ **Example 16.8** I have a rather large bag with a thousand six-sided dice, and a single trick die that always lands on the number 4. If I pick a random die and roll a 4, what is the probability that it is the trick die?

The prior odds can be expressed as $1 : 1000$, and the posterior as $1(1) : 1000(1/6) = 3 : 500$. There is slightly less than an 0.6% chance that I grabbed the trick die, even though it is much more likely to roll a 4 than a regular one. ∎

In many scenarios we may not have access to exact probabilities, but the general principle still applies.

∎ **Example 16.9** You read a news article reporting that a mathematician has claimed to have proved that P $\neq$ NP, solving one of the famous Millennium Problems. How would you go about judging the probability that the mathematician's claim is mistaken? ∎

16.2 Gaussian Mixture Model

Definition 16.2.1 A *Gaussian mixture model* in d with k components is defined by the weights $\{\phi_i\}_{i=1}^k$ and multivariate normal distributions

$$\mathbf{Z}_i = \text{Normal}(\mu_i, \Sigma_i), \quad 1 \leq i \leq k$$

with individual PDFs f_{Z_i}, $1 \leq i \leq k$. The probability density function of the mixture Z is given by

$$f_Z(\mathbf{x}) = \sum_{i=1}^{k} \phi_i f_{Z_i}(\mathbf{x}).$$

One natural interpretation is that the population being described by the mixture model consists of k distinct subpopulations, each described by their own Gaussian distribution. If we assign each subpopulation a categorical label $C \in \{1, \ldots, k\}$, then ϕ_i is the proportion of the whole

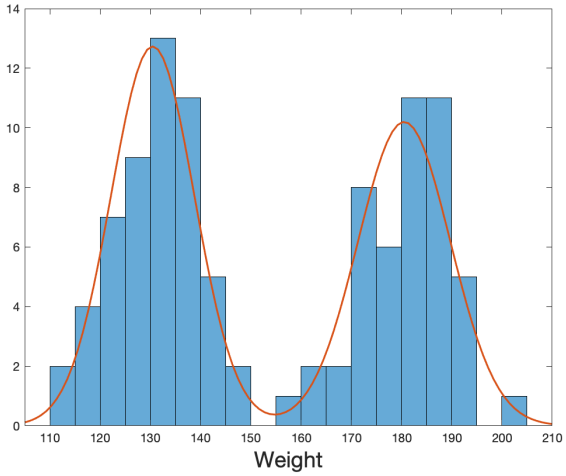


Figure 16.1: Gaussian mixture model with 2 components fit to hospital weight data.

population that falls in subpopulation i, and f_{Z_i} is the probability distribution of Z conditoned on the subpopulation i. In terms of conditional expectations and the Law of Total Probability,

$$f_Z(\mathbf{x}) = \sum_{i=1}^{k} p_C(i) f_{Z|C}(\mathbf{x}|i) = \sum_{i=1}^{k} \mathbb{P}(C=i) f_{Z|C}(\mathbf{x}|C=i).$$

■ **Example 16.10** The `hospital` dataset in Matlab contains weight information for 53 women and 47 men. The women have an average weight of about 130.5 pounds with a standard deviation of about 8.3 pounds. The men have an average weight of about 180.5 pounds with a standard deviation of about 9.2 pounds. Figure 16.1 (generated by Program 16.1) shows a histogram of the patients' weights, along with the approximate distibution resulting from using these parameters (proportions, means, standard deviations) to fit a Gaussian mixture model. ■

It is important to remember that this mixture model, whose PDF is a weighted average of Gaussian PDFs, is *not* the same thing as taking a weighted average of the Gaussian random variables themselves! If we were to define

$$Z = \sum_{i=1}^{k} \phi_i Z_i$$

where $\{Z_i\}_{i=1}^{k}$ are Gaussian variables with a joint Gaussian distribution, then Fact 14.5.3 implies that Z is also just a Gaussian random variable.

16.2.1 Posterior Inference

Given a Gaussian mixture model, we can use a variant of Bayes' Theorem to determine the probability that a given observation belongs to a particular underlying category.

Theorem 16.2.1 — Categorical Posterior Inference. Suppose we have a random variable Z defined by a mixture model with underlying categories $C \in \{1,\ldots,k\}$ and distribution

$$f_Z(\mathbf{x}) = \sum_{i=1}^{k} p_C(i) f_{Z|C}(\mathbf{x}|i).$$

Then given an observation $\mathbf{x}$, the conditional probability that the observation came from a

Program 16.1: Mixture model for hospital data

```
load hospital.mat
BW = 5; % bin width for histogram
N = size(hospital,1); % number of patients
histogram(hospital.Weight,'binWidth',BW), hold on
[mu,sigma] = grpstats(hospital.Weight,...
                hospital.Sex,{'mean','std'});
phi = countcats(hospital.Sex)/N;
Sigma = reshape(sigma.^2,1,1,2); % array of covariance matrices
G = gmdistribution(mu,Sigma,phi);
xs = linspace(105,210);
plot(xs,G.pdf(xs')*BW*N,'LineWidth',1.5) % plot the mixture PDF
xlabel('Weight','FontSize',18)
xlim([105,210])
probs = G.posterior(158);
```

category i is given by the relation

$$p_{C|Z}(i|\mathbf{x}) = \frac{p_C(i) f_{Z|C}(\mathbf{x}|i)}{f_Z(\mathbf{x})}.$$

We could also state this theorem using the "odds" approach:

Fact 16.2.2 The probabilities are given by the relation

$$p_{C|Z}(i|\mathbf{x}) \propto p_C(i) f_{Z|C}(\mathbf{x}|i), \tag{16.1}$$

where $\propto$ means "proportional to".

So to find the probabilities for each i, we can compute the right hand side of 16.1 for each i, then rescale so that the probabilities sum to 1.

■ **Example 16.11** One of the hospital patients from Example 16.10 has a weight of 158 pounds. The odds that the patient is female versus male are

$$p(F)\phi(158;\mu_F,\sigma_F^2) : p(M)\phi(158;\mu_M,\sigma_M^2) \approx (0.53)1.97\cdot 10^{-4} : (0.47)2.15\cdot 10^{-3} \approx 0.094 : 0.906.$$

The mixture model estimates about a 91 percent chance that the patient is male (which the patient is). ■

16.3 Likelihood

Our goal is to turn the problem of sampling from a distribution on its head: given a set of observations, can we determine the distribution from which they were sampled?

This is in general not a simple problem to answer. But if we restrict the set of distributions that we are willing to consider, then we may be able to answer a narrower question: which density function assigns the greatest *likelihood* to the observed data?

Definition 16.3.1 — Likelihood (Discrete). Let p be a probability mass function which depends on a parameter θ. Then given an outcome x, the *likelihood* of θ is defined as the probability of x given θ,

$$\mathscr{L}(\theta|x) = p_\theta(x).$$

For multiple observations $\mathbf{x} = \{x_1, \ldots, \mathbf{x}_n\}$, the likelihood is the product of the individual probabilities.

■ **Example 16.12** Let X be a Bernoulli variable $\text{Ber}(\theta)$, where θ determines the success probability. If we flip a coin 4 times and get 3 heads and 1 tail, then

$$\mathscr{L}(0.5|\mathbf{x}) = \binom{4}{1}(0.5)^3(0.5)^1 = 0.25$$

and

$$\mathscr{L}(0.6|\mathbf{x}) = \binom{4}{1}(0.6)^3(0.4)^1 \approx 0.35.$$

The parameter $\theta = 0.6$ (60 percent heads) assigns a greater likelihood to the observed data. ■

Definition 16.3.2 — Likelihood (Continuous). Let f be a probability density function which depends on a parameter θ. Then given an outcome x, the *likelihood* of θ is defined as the probability density of f at x given θ,

$$\mathscr{L}(\theta|x) = f_\theta(x).$$

For multiple observations $\mathbf{x} = \{x_1, \ldots, \mathbf{x}_n\}$, the likelihood is the product of the individual probability densities. It is common to use the *log-likelihood*

$$\ell(\theta|\mathbf{x}) = \sum_{i=1}^{n} \ln f_\theta(x_i),$$

since the resulting expressions are often simpler to work with.

It is worth noting that in the continuous case the likelihood is not a probability; it is a product of probability densities.

■ **Example 16.13** Let X be a Gaussian variable $\text{Normal}(\mu, \sigma^2)$ where $\theta = (\mu, \sigma^2)$ determines the mean and variance of the distribution. Say we draw two observations, 0 and 1. Then for $\theta_1 = (0, 1)$, we get

$$\ell(\theta_1|\mathbf{x}) = \ln\left(\frac{1}{\sqrt{2\pi}}e^{-0^2/2}\right) + \ln\left(\frac{1}{\sqrt{2\pi}}e^{-1^2/2}\right) = -\frac{1}{2} - \ln(2\pi) \approx -2.34,$$

and for $\theta_2 = (0.5, 1)$ (a mean of 0.5 and unit variance), we get

$$\ell(\theta_2|\mathbf{x}) = \ln\left(\frac{1}{\sqrt{2\pi}}e^{-(0-.5)^2/2}\right) + \ln\left(\frac{1}{\sqrt{2\pi}}e^{-(1-.5)^2/2}\right) = -\frac{1}{4} - \ln(2\pi) \approx -2.09.$$

The parameter θ_2 assigns a greater likelihood to the observed data. ■

The *maximum likelihood estimator* is the parameter θ which assigns the greatest likelihood to the observed data.

Definition 16.3.3 — Maximum Likelihood Estimator. Given a set of observations $\mathbf{x}$ and likelihood function $\mathscr{L}$, the *maximum likelihood estimator* is given by

$$\widehat{\theta}_{\text{mle}} = \arg\max_\theta \mathscr{L}(\theta|\mathbf{x}).$$

Fact 16.3.1 Because $\ln(x)$ increases monotonically with x, the parameter $\widehat{\theta}$ that maximizes $\mathscr{L}(\theta|\mathbf{x})$ is the same as the one that maximizes the log-likelihood $\ell(\theta|\mathbf{x})$.

In a number of situations, the maximum likelihood estimators can be found just by computing the sample mean and sample variance.

Theorem 16.3.2 For a set of observations $\mathbf{x}$ drawn from a Bernoulli distribution $\text{Ber}(\theta)$, the maximum likelihood estimator is exactly the observed proportion of successes,

$$\widehat{\theta}_{\text{mle}} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Proof. Suppose that $\sum_{i=1}^{n} x_i = m$, or that we have m successes out of n trials. For a given θ, the log likelihood is

$$\ell(\theta|\mathbf{x}) = \ln\left(\binom{n}{m}\theta^m(1-\theta)^{n-m}\right) = \ln\binom{n}{m} + m\ln\theta + (n-m)\ln(1-\theta).$$

Taking the derivative with respect to θ gives

$$\frac{d}{d\theta}\ell(\theta|\mathbf{x}) = \frac{m}{\theta} + \frac{m-n}{1-\theta},$$

which is equal to zero precisely when $\theta = m/n$. The second derivative is negative here, so the point is a maximum. ■

The maximum likelihood estimator is only the "most likely" model that fits the data if we start out assuming that all distributions θ are equally likely. If we have other prior beliefs (e.g., that a coin is nearly fair), then those beliefs should still carry some weight even after accounting for new evidence. For example, flipping 3 heads out of 4 does not provide much evidience that a coin is biased, but flipping 30 heads out of 40 is more persuasive. More formally, we might assign some prior probability distribution to θ, and after accounting for new evidence use Bayes' Theorem to determine a posterior distribution on θ. So we would not end up with a single value of θ, but rather a range of values, some of which are more probable than others. But that's a topic for a class on Bayesian inference.

16.3.1 Gaussian Distribution

In the case of fitting a Gaussian to single-dimensional data, we get the very nice result that the maximum likelihood estimator can be found simply by computing the sample mean and sample variance of the data.

Theorem 16.3.3 For a set of observations $\mathbf{x}$ drawn from a Gaussian distribution $\text{Normal}(\mu, \sigma^2)$, the maximum likelihood estimator is given by

$$\widehat{\mu}_{\text{mle}} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \widehat{\sigma}^2_{\text{mle}} = s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

Note the normalization by n rather than $n-1$ for the sample variance, as opposed to the usage in Notation 6.1.

Proof. Given a set of observations $\mathbf{x}$ with sample mean $\bar{x}$ and (biased) sample variance s^2, the log likelihood for a set of parameters $\theta = (\mu, \sigma^2)$ is

$$\ell(\theta|\mathbf{x}) = \sum_{i=1}^{n} \ln\left(\frac{1}{\sigma\sqrt{2\pi}} e^{-(x_i-\mu)^2/(2\sigma^2)}\right) = -\frac{1}{2}n\ln(2\pi) - n\ln\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2.$$

Regardless of σ, this quantity is maximized when $\mu = \bar{x}$; see Fact 12.4.1. For any μ, the partial derivative with respect to σ,

$$\frac{\partial}{\partial \sigma}\ell(\theta|\mathbf{x}) = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i - \mu)^2,$$

is zero precisely when $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$, which when $\mu = \bar{x}$ is equal to s^2. The second derivative shows that this is again a maximum rather than a minimum. ■

An analogous result will hold when fitting a multivariate Gaussian to data in higher dimensions: the MLE is given by the sample mean and covariance matrix

$$\widehat{\mu}_{\text{mle}} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i, \qquad \widehat{\mathbf{K}}_{\text{mle}} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \widehat{\mu}_{\text{mle}})(\mathbf{x}_i - \widehat{\mu}_{\text{mle}})^T.$$

16.3.2 Gibbs' Inequality

Theorem 16.3.2 also has a nice extension to discrete probability distributions. Suppose that we have a probability distribution $P = \{p_1, \ldots, p_n\}$ corresponding to the observed frequencies over n samples. For example, perhaps we have rolled a die with $n = 6$ sides N times, and $p_1, \ldots, p_6$ represent the frequency with which each number has appeared. Then the number of times each number i was rolled is equal to Np_i, and the log-likelihood of observing P given $\theta = Q = \{q_1, \ldots, q_n\}$ is equal to

$$\ln\left(\prod_{i=1}^{n} q_i^{Np_i}\right) = N\sum_{i=1}^{n} p_i \ln q_i.$$

The maximum likelihood estimator is the set of parameters θ that maximimizes this quantity, and a theorem known as Gibbs' Inequality tells us that the unique solution is $\theta = P$. In other words, the underlying frequencies that assign the greatest likelihood to the observed frequencies are precisely the observed frequencies themselves.

Definition 16.3.4 — Information Entropy. For discrete probability distributions P and Q, the *information entropy* of P is defined as

$$H(P) = -\sum_{i=1}^{n} p_i \ln p_i$$

and the *cross entropy* of Q relative to P is defined as

$$H(P,Q) = -\sum_{i=1}^{n} p_i \ln q_i.$$

Theorem 16.3.4 — Gibbs' Inequality. For discrete probability distributions P and Q,

$$H(P) \leq H(P,Q),$$

with equality if and only if $P = Q$.

Proof. Assume for simplicity that $p_i \neq 0$ for all i. Subtracting the left-hand side from both sides and using properties of logarithms, we find that

$$H(P,Q) - H(P) = -\sum_{i=1}^{n} p_i \ln\frac{q_i}{p_i} \geq -\sum_{i=1}^{n} p_i\left(\frac{q_i}{p_i} - 1\right) = -\sum_{i=1}^{n} q_i + \sum_{i=1}^{n} p_i = 0.$$

The inequality uses the fact that $\ln(x) \leq x - 1$ whenever $x > 0$. Since equality holds if and only if $x = 1$, it follows that $H(P) \leq H(P,Q)$ with equality if and only if $q_i = p_i$ for all i, meaning that $P = Q$.

The general proof where $p_i = 0$ for some i is similar, but involves some case handling. ■

16.4 Expectation-Maximization Algorithm

Back to the problem of clustering: one *expectation-maximization* (EM) algorithm for clustering a set of data attempts to find the maximum likelihood estimator among all possible Gaussian mixture models. Like the naive k-means clustering algorithm, it alternates between two steps:

- Given a Gaussian mixture model defined by k Gaussians with proportions ϕ_i and distributions Normal$(\mu_i, \mathbf{K}_i)$, $1 \leq i \leq k$, use Bayes' Theorem (specifically Theorem 16.2.1) to approximate the probability $p(i|\mathbf{x}_j)$ of each point $\mathbf{x}_j$ $(1 \leq j \leq n)$ belonging to each cluster i.
- Refit each cluster independently to the data, weighted by the probabilities $\pi_{ij} = p(i|\mathbf{x}_j)$.

Roughly, the more likely an observation $\mathbf{x}_j$ is thought to belong to a cluster i, the more "responsible" cluster i is for that observation. When the parameters for the ith Gaussian are being recomputed, it is most important to properly explain the observations for which that Gaussian is most responsible.

■ **Example 16.14** With the patient weights from the `hospital` dataset in Example 16.10, Matlab's `fitgmdist` uses the EM algorithm to fit a model by maximum likelihood. When told to find a model with two clusters, it returns components

$$\phi_1 \approx 0.53, \quad \mu_1 \approx 130.5, \quad \sigma_1 \approx 8.3$$

and

$$\phi_2 \approx 0.47, \quad \mu_2 \approx 180.6, \quad \sigma_2 \approx 9.1,$$

which almost perfectly fits the actual summary statistics of the female/male patients. ■

The previous example worked out rather nicely because we had good reason to expect that patient weights might be modeled by two roughly Gaussian distributions with gender as a decisive categorical factor. We can try fitting Gaussian mixture models to other data, but it won't necessarily tell us about any underlying categories.

■ **Example 16.15** Take the `hospital` dataset and fit the patient ages using a mixture model with three components. Running EM will produce a solution, but the mixture model does not clearly correspond to any underlying categories. Figure 16.2 shows the data and fitted model. ■

As with the naive k-means algorithm, there is no guarantee that EM will find the optimal solution. Successive runs of the algorithm may give different outputs depending on the starting points chosen.

■ **Example 16.16** For the Iris dataset, a bad choice of starting parameters (with the default RNG seed, in fact) leads to a poor clustering where two components are fit two the Setosa species (one of which is targeted specificly at just three observations!), and the remaining Gaussian attempts to fit both the Versicolors and Virginicas. Results are shown in Figure 16.2, where the contour plot of the hapless second Setosa cluster appears almost as a thin line. ■

16.4.1 Decision boundaries

The EM algorithm will end up estimating the probabilities π_{ij} that observation j was produced by component i. If we impose some extra structure on the model being fitted, we find that (as with k-means clustering) the decision boundaries will be linear.

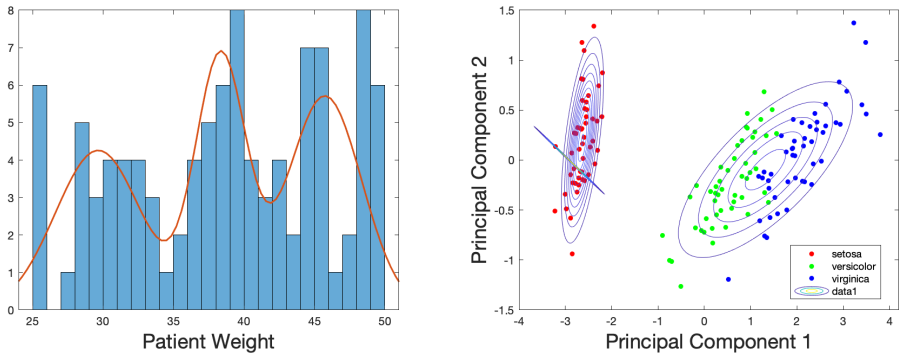


Figure 16.2: Left: mixture model with 3 components for patient weight data. Right: poor results attempting to split the Iris dataset into 3 clusters.

Theorem 16.4.1 — Linear Discriminant Analysis. If all of the components in a mixture model have the same covariance matrix, then all decision boundaries are linear and the region for each cluster is a convex polyhedron.

Proof. Consider two Gaussian components with proportions ϕ_i and distributions Normal$(\mu_i, \mathbf{K})$, $1 \leq i \leq 2$. The decision boundary will be the set where the ratio of the two likelihoods is equal. More generally, we'll weaken this condition and allow the user to set the decision boundary to the set where $p_{C|Z|}(1|\mathbf{x}) = \alpha p_{C|Z}(2|\mathbf{x})$ for some positive number α. From Theorem 16.2.1, equality of the posterior probabilities holds when

$$\phi_1 f_{Z|C}(\mathbf{x}|C=1) = \alpha\phi_2 f_{Z|C}(\mathbf{x}|C=2),$$

which, given the Gaussian distributions of the components, is equivalent to

$$\frac{\phi_1}{\sqrt{(2\pi)^d \det(\mathbf{K})}} e^{-(\mathbf{x}-\mu_1)^T\mathbf{K}^{-1}(\mathbf{x}-\mu_1)/2} = \frac{\alpha\phi_2}{\sqrt{(2\pi)^d \det(\mathbf{K})}} e^{-(\mathbf{x}-\mu_1)^T\mathbf{K}^{-1}(\mathbf{x}-\mu_1)/2}.$$

Take logarithms of both sides, and cancel out the like terms:

$$\ln\left(\frac{\phi_1}{\alpha\phi_2}\right) - \frac{1}{2}(\mathbf{x}-\mu_1)^T\mathbf{K}^{-1}(\mathbf{x}-\mu_1) = -\frac{1}{2}(\mathbf{x}-\mu_1)^T\mathbf{K}^{-1}(\mathbf{x}-\mu_1).$$

Rearranging and distributing the quadratic expressions yields

$$2(\mu_1-\mu_2)^T\mathbf{K}^{-1}\mathbf{x} = 2\ln\left(\frac{\phi_1}{\alpha\phi_2}\right) + \mu_1^T\mathbf{K}^{-1}\mu_1 - \mu_2^T\mathbf{K}^{-1}\mu_2,$$

which is exactly a linear constraint of the form $\mathbf{a}^T\mathbf{x} = \beta$. ■

Things get even simpler if we require the covariance matrices to be multiples of the identity, $\mathbf{K} = \sigma^2\mathbf{I}_d$.

Fact 16.4.2 If all of the components in a mixture model have the same covariance matrix $\sigma^2\mathbf{I}_d$ and equal proportions $\phi_i = \frac{1}{k}$, $1 \leq i \leq k$, then each observation $\mathbf{x}$ is most likely to belong to the cluster with the closest mean μ_i.

In this sense, the "ideal" scenario for k-means clustering is when the data are distributed in spherical Gaussian blobs.

16.5 Matlab Commands

Gaussian Mixture Models

- `fitgmdist(X,k)`: Fits a Gaussain mixture model with `k` components to the data in `X`. See the documentation for `gmdistribution` for the properties and methods associated with this class.

Data Analysis

- `countcats`: Count occurrences of categories in a categorical array's elements.
- `categories`: Get a list of a categorical array's categories.
- `grpstats`: Computes summary statistics by group for data in a table or matrix.

17. Classification

Time to turn from clustering to classification! Now the scenario in mind is that we have some amount of labeled data on which to *train* a model, and some amount of unlabeled data whose labels we want to use our model to predict. In the abstract, we train the model on the set of data

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \quad \mathbf{x}_i \in \mathbb{R}^d, \quad y_i \in \{1, \ldots, m\},$$

where n is the number of data points, d is the number of dimensions, and m is the number of class labels. The x-data represent the *observations* and can be taken as a single data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, and the y-data represent the categories for those observations and can be taken as a single vector $\mathbf{y} \in \mathbb{R}^n$. A classifier can then be defined as a function

$$f : \mathbb{R}^d \to \{1, \ldots, m\},$$

which takes observations as input and returns the predicted class label as output.

In this chapter we'll examine two types of classifiers:

- KNN classification, or k nearest neighbors, has some similarity to single-linkage clustering in that it uses local information to classify new points and can easily handle data for which nonlinear decision boundaries are needed;
- Linear least squares classification solves a least squares problem to determine its classification model, and the resulting model will necessarily have linear decision boundaries.

We will start by discussing a very simple method based on k-means clustering. It isn't very sophisticated, but we mention it in order to get a sense of some of the costs and considerations involved.

1. Compute the means $\mu_j \in \mathbb{R}^d$ for each class $1 \leq j \leq m$.
2. Classify an unknown point $\mathbf{x}$ by giving the class of the closest mean μ_j.

A few of the basic properties this algorithm has:

- Due to the curse of dimensionality, dimension reduction should probably be applied first if d is large.

- The model is sensitive to the scaling of the data.
- Its decision boundaries will necessarily be linear.
- The means are cheap to compute (total cost $\mathcal{O}(nd)$).
- Classifying data is also cheap (cost $\mathcal{O}(md)$ per observation).
- The model is cheap to update if new observations are added, since the means do not need to be recomputed from scratch.

This last point is one that we have not mentioned previously, but depending on the application it can be extremely useful. In situations where the data set is continuously being updated (new books at a bookstore or new movies on Netflix, and generally called *online learning*), we would like to avoid having to solve an expensive problem to retrain the model each time a new observation is added. In general, different classification strategies may involve tradeoffs between the cost of *training* and/or *updating* the model, versus the cost of *using* the model to classify unknown observations. The k-means strategy discussed just now is very cheap on both fronts, but it is also not very sophisticated.

Fact 17.0.1 If we define the means of two sets

$$\mu_x = \frac{1}{n_x}\sum_{i=1}^{n_x} x_i, \qquad \mu_y = \frac{1}{n_y}\sum_{i=1}^{n_y} y_i,$$

then the mean of the union of the sets is given by

$$\hat{\mu} := \frac{1}{n_x+n_y}\left(\sum_{i=1}^{n_x} x_i + \sum_{i=1}^{n_y} y_i\right) = \frac{1}{n_x+n_y}(n_x\mu_x + n_y\mu_y).$$

17.1 K Nearest Neighbors

The strategy for k nearest neighbors classification is exactly what it says on the tin:

1. For a new observation $\mathbf{x}$ that needs classification, find the k nearest neighbors $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$ in the training set.
2. Find the classes $y_{i_1}, \ldots, y_{i_k}$ associated with those neighbors, and assign to $\mathbf{x}$ whichever class gets a plurality (or majority, if there are only two classes).

In the case of binary classifcation (where there are only $m = 2$ classes), it's typical to make k an odd number to avoid ties. Otherwise, you'll want to pick some sort of tiebreaking strategy.

When implementing k-nearest neighbors, we face a few of the same design choices as when we were looking at clustering problems. First, we need to decide how to choose k, since different values of k can result in different decisions about how a given point should be classified (see Figure 17.1). Larger values of k generally lead to smoother decision boundaries and reduce the effect of outliers, but setting k too large may give too much weight to distant data points. One possible strategy is to look at a weighted average of the k nearest neighbors, where the closest neighbors are weighted more heavily.

Second, if the data is high-dimensional (say $d > 10$) then it is a good idea to carry out dimension reduction first, particularly if we intend to use the Euclidean distance. Third, the data should be scaled appropriately so that differences along one particular coordinate axis are not given too much weight relative to the others (see Figure 17.2).

One of the main advantages of KNN classification is its ability to handle nonlinear decision boundaries. Some of its other attractive properties relate to the costs involved in training the model.

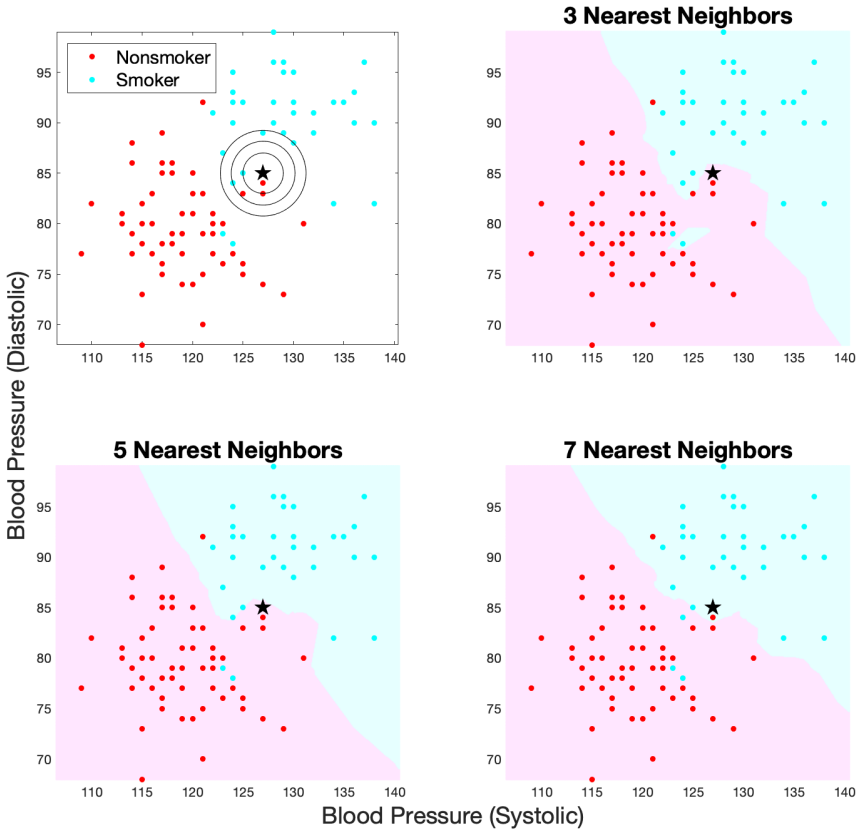


Figure 17.1: Predicting whether a hospital patient is a smoker or non-smoker based on blood pressure using KNN with $k = 3, 5, 7$. Decision regions generally become smoother as k increases.

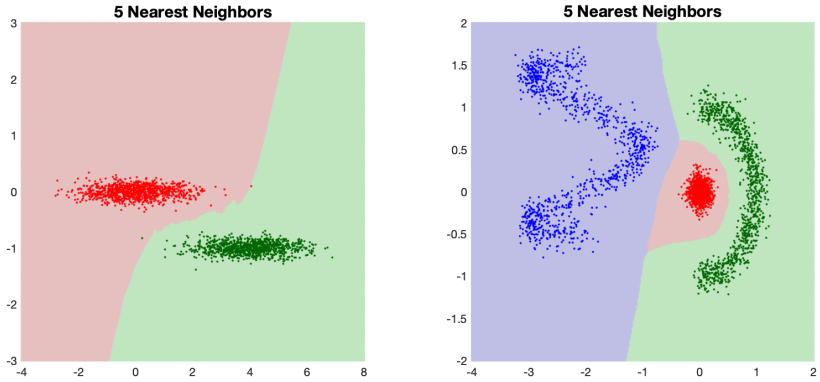


Figure 17.2: Left: Normalize the data for better results. Right: KNN allows for nonlinear decision boundaries.

17.1.1 Lazy Learning

KNN is an example of what's known as a *lazy learning* algorithm, which essentially means that the "training" phase of the algorithm does not involve any actual construction of a classifying function f. Generally, these types of algorithms will be cheaper to train and update, but classifying new observations will be more expensive.

With the naive approach: the training phase involves no work at all. To classify a new observation $\mathbf{x}$, we have to compute the distance between $\mathbf{x}$ and each point $\mathbf{x}_i$ in the training set, for a cost of $\mathcal{O}(nd)$. From that point, finding the k nearest neighbors is relatively cheap.

By doing some work ahead of time, it is possible to avoid having to compare $\mathbf{x}$ to *every* training point in order to find the k nearest neighbors. One strategy is to organize the data into a *k-d tree*, a generalization of a binary tree to higher dimensions. Building the tree can cost $\mathcal{O}(dn\log n)$, but classifying new points will cost only $\mathcal{O}(\log n)$ on average.

Other strategies for reducing the cost of new queries include the following:

- If you are running a database with a large number of users and certain queries are very common, compute the results for those common queries ahead of time to avoid having to run the search from scratch.
- Use faster search methods to find the *approximate* k nearest neighbors rather than the exact k.
- When the training set is very large, reduce the size of the data set in a manner that roughly preserves the decision boundaries. This may involve removing outliers, or eliminating redundancy by removing points that are near a large number of other points with the same class label.

17.2 Linear Least Squares Classification

At the other end of the spectrum of classification algorithms we can find least squares classification: a method that does a lot of work to train the model and cannot be updated as easily, but allows for extremely cheap classification. We'll start by covering the case of *binary classification*, and then generalize from there to allowing more than two class labels.

On its own, binary classification has a wide variety of uses, particularly when some agent is faced with making a binary decision:

- Spam blockers: should a particular email or phone call be flagged as junk?
- Disease testing: does a given person have Covid?
- Credit fraud: should the credit card company notify a card holder about a potentially suspicious purchase?
- Quality control: does a given product pass specifications?
- Chick sexing: can a hatchery efficiently separate the male chicks from the female?
- Facial/fingerprint recognition: should your phone unlock for the user or not?

For binary classification, the class labels will be switched to ± 1 values,

$$y_i \in \{+1, -1\}, \quad 1 \leq i \leq n,$$

where $+1$ is the *positive* outcome and -1 the *negative*. Typically, the positive outcome is the category "of interest" if there is such a distinction to be made: positive Covid tests and instances of credit card fraud are relatively rare, for example, but we would like to make sure that we do identify those cases when they occur.

In any case, the strategy we'll adopt is to set up the classification problem in terms of a least squares problem: for observations $\{\mathbf{x}_i\}_{i=1}^n$ and labels $\{y_i\}_{i=1}^n$, we would ideally like to find an linear

classification function

$$y_i = f(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_d x_{id} = \beta^T \mathbf{x}_i', \qquad \mathbf{x}_i' = \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}, \qquad 1 \le i \le n.$$

If we have many more data points than dimensions ($n \gg d$), the equation will be overdetermined and highly unlikely to have an exact solution. So instead we'll give ourselves the least squares problem

$$\min_{\beta} \sum_{i=1}^{n} (\beta^T \mathbf{x}_i - y_i)^2 = \min_{\beta} \|\mathbf{X}'\beta - \mathbf{y}\|_2^2,$$

where

$$\mathbf{X}' = \begin{bmatrix} \mathbb{1} & \mathbf{X} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d}^T \\ 1 & x_{21} & \cdots & x_{2d}^T \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{bmatrix} \in \mathbb{R}^{n \times (d+1)}, \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}.$$

Once we have found the optimal coefficients β, we can obtain the predicted values for the testing data $\{x_i^{\text{test}}\}_{i=1}^{n_{\text{test}}}$,

$$\tilde{y}_i = \beta^T x_i^{\text{test}}, \qquad 1 \le i \le n_{\text{test}},$$

but face the slight wrinkle that these values are probably not going to be equal to $+1$ or -1, and so aren't themselves valid class predictions. No matter—we'll simply round them by setting a threshold τ, and making the predictions

$$\hat{y}_i = \begin{cases} +1 & \tilde{y}_i > \tau, \\ -1 & \tilde{y}_i < \tau. \end{cases}$$

Cases where $\tilde{y}_i = \tau$ are indeterminate and can be rounded either way. By default we can just use $\tau = 0$, in which case we would have $\hat{y}_i = \text{sign}(\tilde{y}_i)$, but we'll see shortly that there may be good reasons for choosing other values.

■ **Example 17.1** Consider the 1-dimensional data set

$$\{\mathbf{x}_i\} = \{-45, -24, -15, -10, -6, 6, 10, 15, 17, 29, 44, 48\},$$
$$\{y_i\} = \{-1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1\}.$$

Generally, larger values of x are associated with the positive label, but there isn't a clean split. Setting up and solving the least squares problem yields the coefficients

$$\beta_0 \approx -0.128, \quad \beta_1 \approx 0.0223.$$

If we have the two unlabeled observations $\{0, 35\}$, then the resulting predicted values are

$$\tilde{y}_1 \approx -0.13, \qquad \tilde{y}_2 \approx 0.65,$$

which with a threshold of $\tau = 0$ respectively get rounded to $\hat{y}_1 = -1$ and $\hat{y}_2 = 1$. Figure 17.3 shows the results of this process. ■

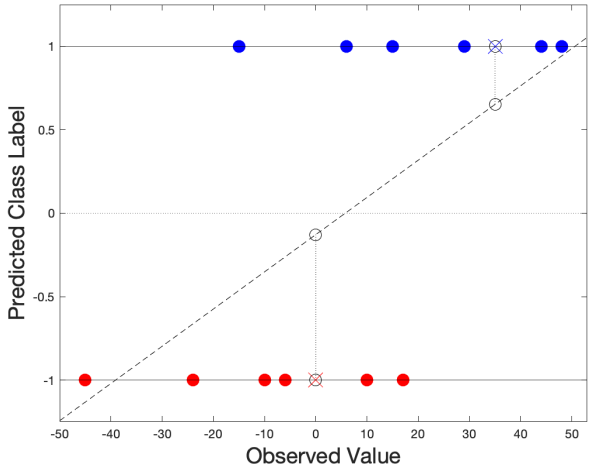


Figure 17.3: Linear classification for one-dimensional data.

17.2.1 Eager Learning

Unlike KNN, linear classification takes quite a bit of work to train the model. It has to solve a least squares problem of size $n \times (d+1)$, which requires $\mathcal{O}(nd^2)$ computation. Neither is the model trivial to update when new observations are added, although there are more efficient methods than solving the whole problem from scratch. However, once the model has been trained and the optimal coefficients β obtained, it is extremely cheap to classify new points: computing $\beta^T \mathbf{x}$ requires only $\mathcal{O}(d)$ operations! Linear classification can therefore be useful in situations where we want to classify lots of new points, but do not need to update the model very often.

17.2.2 Training and Testing

Let's see how the linear classifier works on some of the MNIST data. We'll consider the digits 3 and 8 (which should be relatively difficult to distinguish as pairs of digits go), and start by taking the top two principal components for linear classification. Interestingly, the combination of the top two principal components appears to be much more useful than either one individually (unlike the 0/1 example from Section 11.3.4).

It's important to bear in mind that this linear classification strategy does not assign *probabilities* to any of its guesses. However, a reasonable interpretation of the scores $\tilde{y}_i$ is that larger values generally correspond to observations that are more likely to belong to the $+1$ class. The bottom left plot of Figure 17.4 shows a resulting histogram of the predicted scores $\tilde{y}_i$, grouped by the true class labels. The threshold τ corresponds to a cutoff point: everything to its right will be predicted to belong to the class $+1$, and everything to the left will be predicted to belong to the class -1. We can get different accuracy rates by adjusting τ, but since the two histograms overlap there is no value of τ that can give us 100 percent accuracy. In this specific example, the cutoff $\tau = 0$ yields an accuracy rate of about 85.9 percent on the training data and about 86.8 percent on the testing data.

Always be sure to test the model on data that does overlap with the data used to train the model. If the goal is to allow the model to make new predictions, we must evaluate its ability to classify *unseen* data, not data for which the labels are used to fit the model in the first place. The primary risk that we wish to avoid is *overfitting* the training data: ultimately, we want to find a model that is a good fit to the underlying distribution of the data rather than the specific idiosyncracies of the sample used for training. For this example with classifying the digits 3 and 8, using the top 10 principal components rather than just 2 increases the training accuracy to about 93.5 percent and

Program 17.1: Multiclass Digit Classification

```
[COEFF,SCORE] = pca(trainX,'numComponents',2); % PCA on training data
% INCORRECT
[CTEST,STEST] = pca(testX,'numComponents',2); % CTEST =/= COEFF
% CORRECT
STEST = (testX-mean(trainX))*COEFF; % same reduction as for training data
```

the testing accuracy to about 94.6 percent. But how do we know the proper number of components to choose? Pick too many, particularly if some observations are strongly correlated, and the matrix $\mathbf{A}$ for the least square problem may become ill-conditioned and overly sensitive to the training data. So bear in mind that just because a model performs well on the training data does not necesarily mean that it will generalize well.

It may be tempting to choose the optimal number of components (and other possible parameters that go into the model) based on how well the trained model performs on the testing data. Strictly speaking, this is not proper practice either—from the statistical point of view, you only get one shot at the testing data set. One strategy for parameter tuning is to use *cross-validation*, where the training data is split into k parts, and the model is trained on $k-1$ of the parts and tested on the remaining one. Repeat for each of the k choices for which part of the data will be reserved for testing, observe the performance, and tune the parameters for the model accordingly.

On the topic of dimension reduction, make sure that you analyze the training and testing data using the same coordinate system! If you run PCA for the training and testing data separately, the principal components for the two sets of data will not be equal, so a model that works for the top principal scores of the training data would not necessarily work on the top principal scores for the testing data.

17.3 Evaluation

So what is the best way to set the threshold τ? The most straighforward choice is to set it to maximize the number of correct predictions and minimize the number of incorrect ones. When distinguishing handwritten digits or Iris species, this is probably the most reasonable approach. For other problems, however, not all errors are equally damaging. Given the choice between the two, the designers of Covid test would probably rather have it report that a person does have Covid even if they do not (a *false positive*, or *type I error*) than have it report that a person does not have it when they acutally do (a *false negative*, or *type II error*). It is impossible to eliminate both types of error, but by adjusting τ we can choose a balance between the two appropriate for the situation.

17.3.1 Confusion Matrix

The final plot in Figure 17.4 shows a *confusion matrix* for the predicted class labels versus the actual values on the testing data. The outcomes fall under one of four categories:

- *True positives* (TP) are when the predicted and acutal label are both $+1$;
- *True negatives* (TN) are when the predicted and acutal label are both -1;
- *False positives* (FP) are when the predicted label is $+1$ but the actual is -1;
- *False negatives* (FN) are when the predicted label is -1 but the actual is $+1$;

The accuracy rate is then given by the number of correct guesses divided by the total,

$$\text{accuracy} = \frac{\text{TP}+\text{TN}}{\text{total}},$$

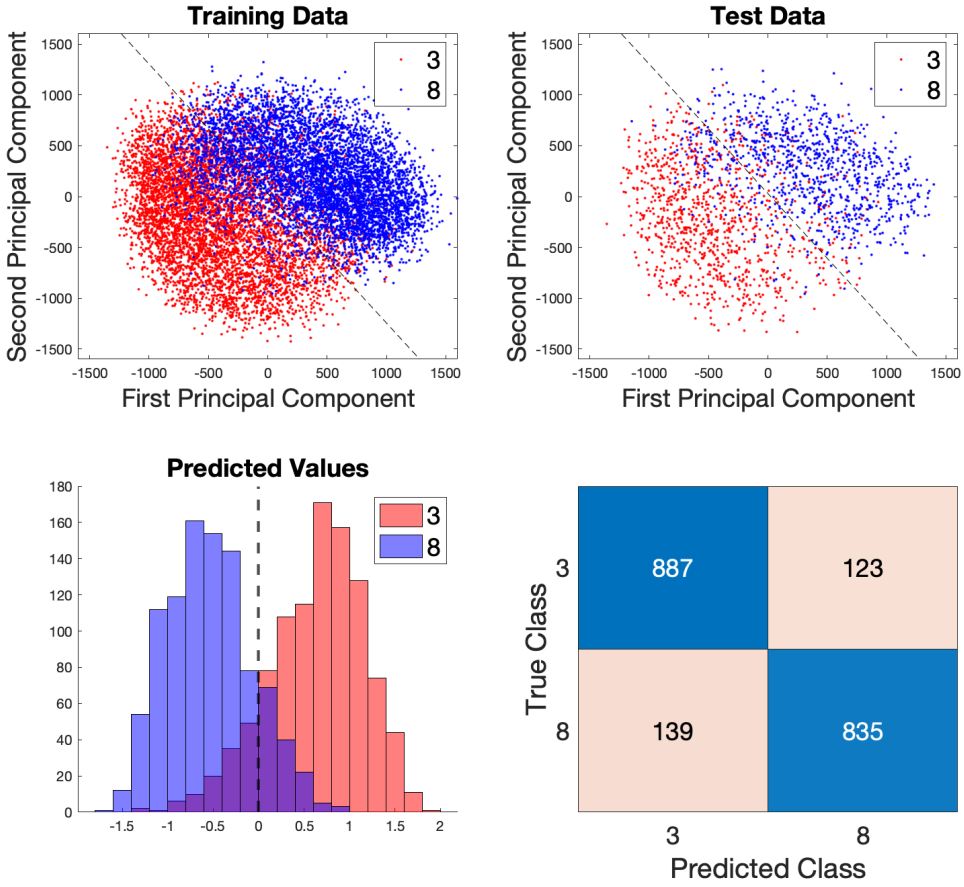


Figure 17.4: Linear classification for distinguishing the handwritten digits 3 and 8.

and the *error rate* is its complement, the number of incorrect guesses divided by the total:

$$\text{error rate} = \frac{\text{FP} + \text{FN}}{\text{total}}.$$

But as mentioned, we might care about other measurements of quality besides pure accuracy. This is particularly the case if the positive class label is rare compared to the overall population.

■ **Example 17.2** A system for predicting winning lottery tickets that guesses that every ticket is a loser will be right well over 99 percent of the time, but completely useless. ■

Perhaps a better measure of a lottery ticket predictor is how many winning tickets it successfully identified?

■ **Example 17.3** A model that guesses that every lottery ticket is a winner will successfully identify every actual winner. It will also be completely useless. ■

Okay, that's not quite right either. Relatedly, the economist Paul Samuelson once joked that "the stock market has predicted nine of the past five recessions." So what we *really* want to know is this: if we buy every lottery ticket that our model predicts to be a winner, will we be able to turn a profit? But how can we measure this based on the information from the confusion matrix?

Definition 17.3.1 — Evaluation Metrics. Given a set of predicted and actual class labels for a binary classifier,

- The *true positive rate*, *sensitivity*, or *recall rate* is the proportion of positive cases caught by the test,

  $$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

  Its complement is the *false negative rate*.
- The *true negative rate*, or *specificity*, is the proportion of negative cases successfully labeled negative by the test,

  $$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}.$$

  Its complement is the *false positive rate*.
- The *positive predictive value*, or *precision*, is the fraction of cases predicted to be positive which are in fact positive,

  $$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

  Its complement is the *false discovery rate*.
- The *negative predictive value* is the fraction of cases predicted to be negative which are in fact negative,

  $$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}}.$$

  Its complement is the *false omission rate*.

■ **Example 17.4** Figure 17.5 shows some possible outcomes in a situation where the digit 8 appears significantly more frequently than the digit 3. In the first scenario, we set $\tau = 0.75$ so that very few 8's are correctly identified as 3's.

- The recall rate is 43.0 percent. We have positively identified fewer than half of the threes.

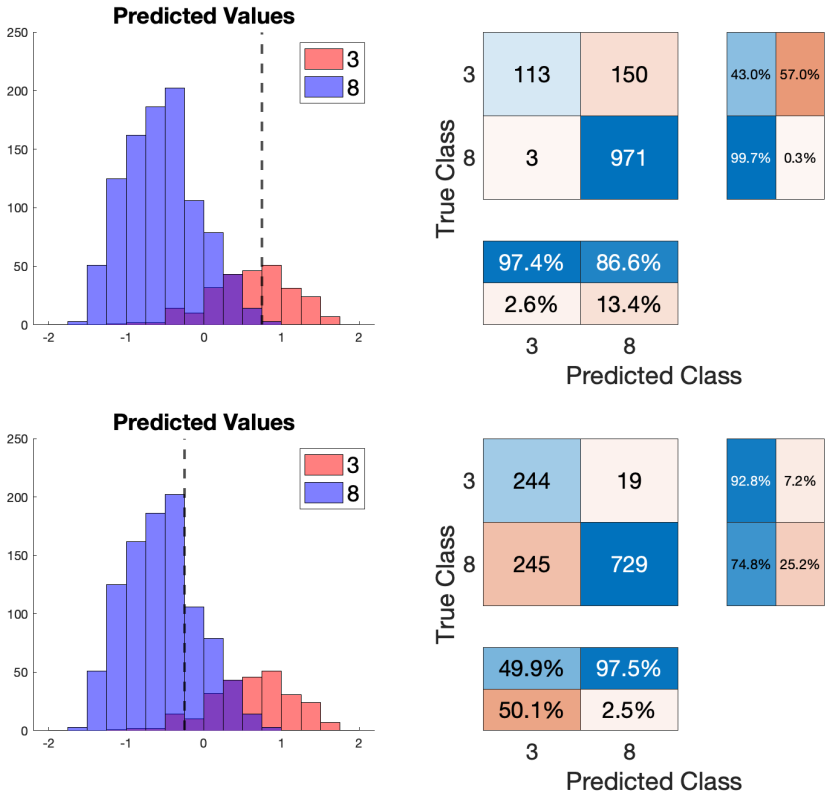


Figure 17.5: Performance metrics for two different thresholds τ. Larger values of τ have higher false positive rates but lower false negative rates.

- In exchange, the precision is quite good. Out of the 116 digits identified as threes 113 were correct, for a rate of 97.4 percent.
- Out of 974 eights, only 3 were identified as threes, for a false positive rate of a mere 0.3 percent.

In the second scenario, we set $\tau = -0.25$ in order to catch most of the 3's.

- The recall rate is now much better: 244 of the 263 threes were successfully identified, for a rate of 92.8 percent.
- The precision has suffered: out of the 489 digits identified as threes, only 244 (a slight minority!) actually are, for a rate of 49.9 percent.
- The false positive rate is also much higher, with 25.2 percent of eights being incorrectly identified as threes.

■

With digit identification we probably don't have much reason to prefer one scenario over the other. With other applications such as credit card fraud detection, we might!

17.3.2 Performance Curves

If we haven't yet decided how to set the threshold τ, one application-neutral way to evaluate the performance of the classifier is to plot a curve illustrating the tradeoffs over *all* possible values of τ. There are many possible ways to do this, but the two we'll mention in particular are the *ROC* curve (short for *receiver operating characteristic*), and the *precision-recall* curve.

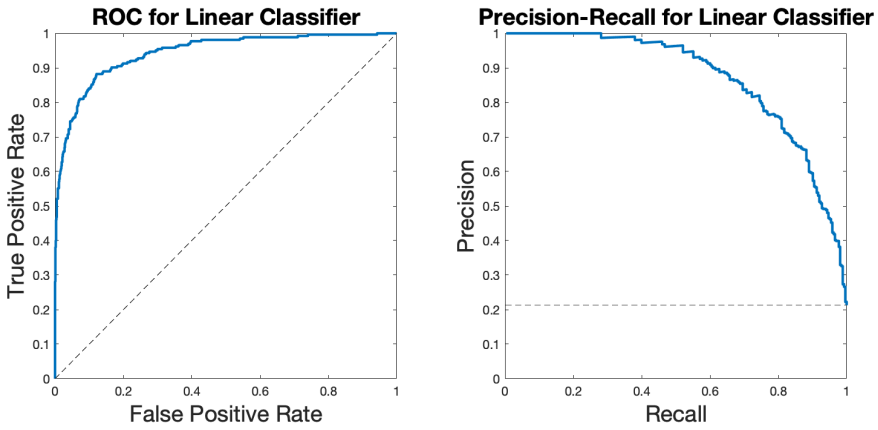


Figure 17.6: ROC and precision-recall curves for digit classification. Dashed lines indicate values attainable by guessing randomly.

Definition 17.3.2 Given a set of true binary class labels $\{y_i\}_{i=1}^n$ and scored values $\{\tilde{y}_i\}_{i=1}^n$, the ROC curve plots the false positive rate on the x-axis and the true positive rate (aka sensitivity or recall) on the y-axis.

Definition 17.3.3 Given a set of true binary class labels $\{y_i\}_{i=1}^n$ and scored values $\{\tilde{y}_i\}_{i=1}^n$, the precision-recall curve plots the recall rate (aka TPR) on the x-axis and the precision (aka positive predictive value) on the y-axis.

One important point that the Example 17.4 illustrates is that when the positive cases are rare compared to the overall population, it is possible to have good accuracy, true positive, *and* false positive rates, but poor precision. Thus when the positive and negative populations are highly imbalanced, the precision-recall curve may give a better indication of model performance than the ROC curve.

In any case, both curves for the digit classification problem are shown in Figure 17.6. Any point along these curves is attainable by the model by setting τ appropriately. For example, the flat line at the top of the precision-recall curve suggests that we can get a recall rate of 27 percent (identify 27 percent of threes) with a precision of 100 percent (every observation identified as a three is in fact a three). Or if we want a better recall rate, we could reach a recall of 88 percent at the cost of having the precision drop to 66 percent.

Dashed lines indicate values attainable by guessing randomly: in the case of the ROC curve, randomly guessing positive labels with probability π will yield both true positive and false positive rates of π. For the precision-recall curve randomly guessing positive labels with probability π will (as before) yield a true positive rate (recall) of π, but the precision will be given by the base rates of the respective populations:

$$\text{PPV} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \frac{\pi \cdot \text{P}}{\pi \cdot \text{P} + \pi \cdot \text{N}} = \frac{\text{P}}{\text{P}+\text{N}}.$$

Thus as π increases, the recall will tend to 1 (we successfully identify all of the positive cases) but the precision will stay the same (out of the observations *identified* as positive, the proportion which are actually positive is the same as for the overall population). One simple way to measure the quality of the model is to compute the area under one of these performance curves.

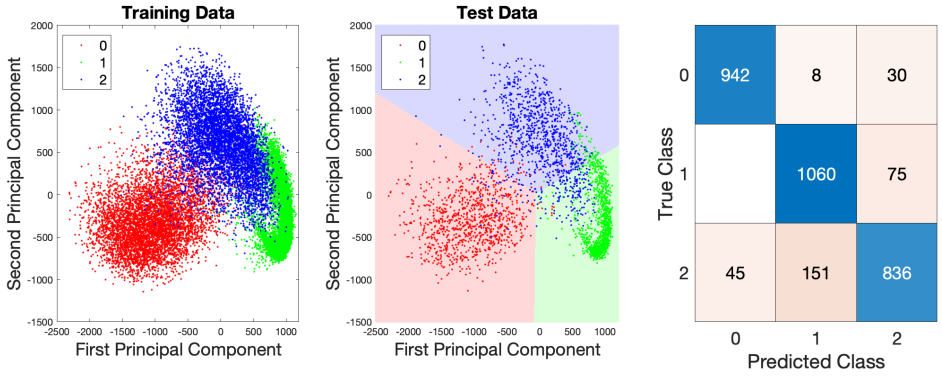


Figure 17.7: Multiclass Digit Classification

17.3.3 Multiclass Classification

What if we have more than two classes, as for the digits 0-9 or the three species of Iris? One simple approach using linear regreesion is that if there are m classes, then we solve m least squares problems. For each class X we can fit a linear model to the binary classification problem of determining X $(+1)$ versus not-X (-1), solving the *least-squares problem with multiple right-hand sides*

$$\min_{\mathbf{X}} \|\mathbf{AX}-\mathbf{B}\|_F^2 = \sum_{i=1}^{m} \min_{\mathbf{x}_i} \|\mathbf{A}\mathbf{x}_i - \mathbf{b}_i\|_2^2.$$

If a direct approach is used (e.g. finding a factorization of $\mathbf{A}$ such as the SVD), then solving this combined system will still cost only $\mathcal{O}(nd^2)$ operations for n data points in d dimensions. By contrast, solving each of the m systems individually from scratch would cost $\mathcal{O}(mnd^2)$, so just let Matlab's backslash handle the work!

In any case, we will now end up with m different linear models, assigning to each point the set of scores

$$(f_1(\mathbf{x}),\ldots,f_m(\mathbf{x}) = (\tilde{y}^{(1)},\ldots,\tilde{y}^{(m)}).$$

After possibly adjusting each value by a set of thresholds $(\tau_1,\ldots,\tau_m)$, we can simply choose the label with the highest assigned score. Figure 17.7 shows the results for classifying the digits 0, 1, and 2 based on the top two principal components of the training data.

Interestingly, by multiplying the fitted coefficients for our linear models by the principal compoents, we can get an interpretation of the linear classifier in terms of the original pixels. Each pixel carries a certain amount of "evidence" for or against each digit, and the total evidence for each digit from the entire image is equal to the sum of the evidences over each pixel. At the end, the assigned label is the one with the most evidence in its favor (up to shifting by the tolerances τ, if desired). Figure 17.8 gives a visual for this interpretation, produced by the code in Program 17.2.

17.4 Logistic Regression

While greater scores $\tilde{y}$ in the linear classification method generally suggest a greater probability that $y = 1$, it is not at all clear exactly how the probabilities change with the scores. There is little difference on an absolute scale between 10^{-2} and 10^{-6}, but in probabilistic terms the difference between a 1 in 100 chance of something happening and a one in a million chance is massive. If

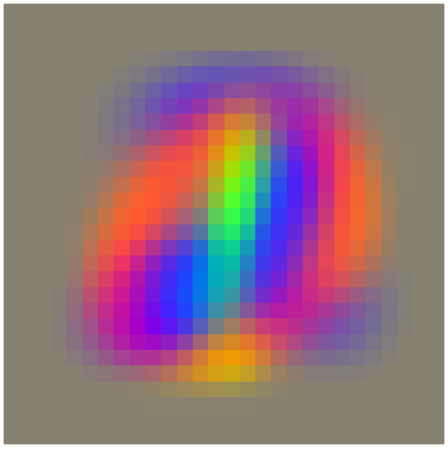


Figure 17.8: Red pixels carry the strongest “evidence” in favor of a handwritten digit being a zero, and similarly for green (1) and blue (2).

Program 17.2: Multiclass Digit Classification

```
load mnist.mat
trainX = double(trainX(trainY<=2,:));
trainY = trainY(trainY<=2)';
[COEFF,SCORE] = pca(trainX,'numComponents',2);
B = [trainY==0, trainY==1, trainY==2]; % Set up and solve LS problem
A = [ones(size(SCORE,1),1),SCORE];
C = A\B;
P = C(1,:) + COEFF*C(2:end,:); % interpretation of coefficients
P = reshape(P,28,28,3);
for i = 1:3
    M = P(:,:,i);
    mx = max(M,[],'all');
    mn = min(M,[],'all'); % Scale to make the colors more visible
    M = (M - mn)/(mx -mn);
    P(:,:,i) = flipud(rot90(M));
end
imshow(P,'InitialMagnification','fit')
```

Figure 17.9: Probabilistic classification of an image. Source: Twitter [19]

such nuance is desired for a given application, we might consider adopting a *probabilistic classifier*: here, the classifier for m classes can be represented as a function $f : \mathbb{R}^d \to [0,1]^m$ such that

$$f(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) & f_2(\mathbf{x}) & \cdots & f_m(\mathbf{x}) \end{bmatrix},$$

where $f_i(\mathbf{x})$ is the estimated probability that the observation has class label i (i.e., $y = i$). The function has to be a discrete probability distribution, so $\sum_{i=1}^m f_i(\mathbf{x}) = 1$ necessarily. If we have to predict a single class label, then we can set y to take on the value i for which $f_i(\mathbf{x})$ is maximized.

One conceptually straightforward approach is to fit a Gaussian mixture model (Section 16.2) to the training data, from which point the probabilities assigned to the testing data would be given by Theorem 16.2.1. Theorem 16.4.1 establishes that if all of the class distributions have the same covariance matrix, then the decision boundaries that result from expectation-maximization will be linear. This result was one motivation for the linear classifier we introduced in the previous section, but it also motivates a probabilistic classifier known as *logistic regression*. To start, we can extend Theorem 16.4.1 in the following manner.

Theorem 17.4.1 — Gaussian Posterior Odds (General). Consider a Gaussian mixture model with m components in d dimensions such that all components have the same covariance matrix. Then there exist vectors $\beta_1, \ldots, \beta_m \in \mathbb{R}^{d+1}$ such that the posterior odds for each class fall in the proportions

$$p(y = 1|\mathbf{x}) : p(y = 2|\mathbf{x}) : \cdots : p(y = m|\mathbf{x}) = e^{\beta_1^T \mathbf{x}} : e^{\beta_2^T \mathbf{x}} : \cdots : e^{\beta_m^T \mathbf{x}}.$$

This assumes that the observations are represented as $\mathbf{x} = [1, x_1, \ldots, x_d] \in \mathbb{R}^{d+1}$, where the leading 1 allows for a constant coefficient.

In the special case of binary classification, the statement can be simplified by the fact that $p(y = -1) = 1 - p(y = 1)$.

Theorem 17.4.2 — Gaussian Posterior Odds (Binary). Consider a Gaussian mixture model with 2 components in d dimensions such that all components have the same covariance matrix. Then there exists a vector $\beta \in \mathbb{R}^{d+1}$ such that the posterior odds for the two classes fall in the

proportions

$$p(y=-1|\mathbf{x}) : p(y=1|\mathbf{x}) = 1 : e^{\beta^T \mathbf{x}}.$$

Equivalently,

$$p(y=1|\mathbf{x}) = \frac{e^{\beta^T \mathbf{x}}}{1+e^{\beta^T \mathbf{x}}} \quad \text{and} \quad p(y=-1|\mathbf{x}) = \frac{1}{1+e^{\beta^T \mathbf{x}}}.$$

17.4.1 Maximizing Likelihood

One motivation for logistic regression is the insight that we can entirely cut out the step of fitting a Gaussian mixture model, and go straight to finding the parameters $\beta_1, \ldots, \beta_m$ that maximize the likelihood of the training data. There isn't a closed-form solution for this problem, and so in practice various iterative algorithms are used. In order to avoid overfitting the data, a variety of regularization strategies can also be employed.

A few common definitions used in this context:

Definition 17.4.1 The standard *logistic* function is defined over $\mathbb{R}$ as

$$\text{logistic}(t) = \frac{e^t}{1+e^t}.$$

Definition 17.4.2 The *logit* function is the inverse of the logistic function,

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right).$$

Binary logistic regression in one dimension can then be said to use the model

$$\mathbb{P}(y=1|x) = \sigma(\beta_0 + \beta_1 x).$$

Definition 17.4.3 The *softmax* function is defined as

$$\text{softmax}(k, x_1, \ldots, x_m) = \frac{e^{x_k}}{\sum_{i=1}^m e^{x_i}}.$$

The softmax function has behavior similar to finding the maximum of the arguments $x_1, \ldots, x_m$ (if one x_k is much larger than the rest then the function will be nearly equal to 1 at that point and nearly zero at the others), but has the added advantage of being differentiable.

■ **Example 17.5** Consider the same one-dimensional dataset that was used in Example 17.1. Figure 17.10 shows the predicted probabilities of a point belonging to one category or another, with the fitted model being

$$p = \text{logistic}(-0.3878 + 0.0651x).$$

The point $x = 0$ is predicted to have a 40 percent chance of having class label $y = 1$, and the point $x = 35$ is predicted to have an 87 percent chance of having class label $y = 1$. ■

17.4.2 Elo Ratings

One application closely related to the logistic model is the *Elo rating system* used for evaluating players in chess and other games. This could be though of as a classification problem in the sense

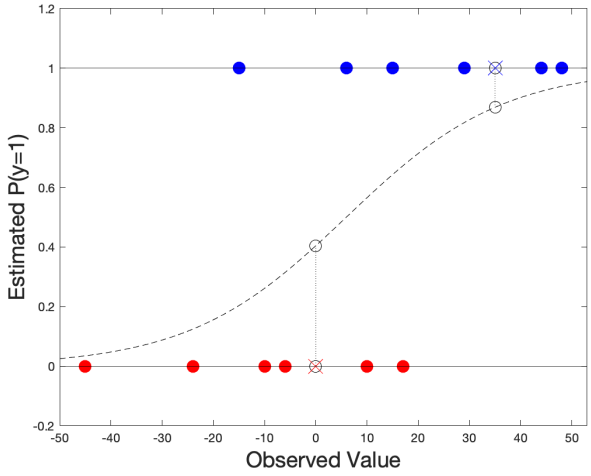


Figure 17.10: Logistic regression on a one-dimensional data set.

that given two players A and B with ratings x_A and x_B, the model would like to predict which player will win a given game. If E_A is the expected number of points A will win (0 for a loss, 1 for a win, possibly 1/2 for a tie), then the Elo system uses the model

$$E_A : E_B = 10^{R_A/400} : 10^{R_B/400},$$

which translates to the expected score

$$E_A = \frac{1}{1+10^{(R_B-R_A)/400}}.$$

This implies that the raw scores of the players do not matter; only the *absolute difference* of their scores is considered in predicting the outcome of the game. It also means that the chance of the weaker player getting an upset victory decreases exponentially with respect to the difference in their ratings: a player with a rating 100 points lower would have an expected score of about 0.36, but a player with a rating 600 points lower would have an expected score of only 0.03.

In theory, one could collect the data on all games people have played, then use maximum-likelihood estimation to determine everyone's ratings. In practice, this faces a variety of problems:

- Players' skills change over time, especially since new players can be expected to improve rapidly.
- Players have some ability to choose their opponents, and might deliberately target opponents who they think are overrated or avoid difficult/underrated ones.
- Lack of transparency: players like being able to know or predict how their rating will change when they win or lose a game!

In practice, the Elo system uses a simple updating rule: for some value K representing the value of the game in points, Player A's score will update as $x_A^{(\text{new})} = x_A + K(y_A - E_A)$, and Player B's will update similarly. Thus games have a zero-sum effect on the players' ratings, and the effect will be greater the more of an upset the outcome is.

■ **Example 17.6** Suppose $K = 30$ and Player A has an expected score $E_A = 0.4$. If A loses, then their rating will drop $30(0.4) = 12$ points and B's rating will increase 12 points. If A wins, their rating will go up $30(0.6) = 18$ points and B's rating will drop 18 points. ■

The updating rule is subject to some argumentation and various tweaks (e.g., what is a good value of K?) but it has the advantage of being simple enough that players can figure out for themselves how their rating will be affected by their performance in a tournament.

17.5 Matlab Commands

Evaluation

- `confusionmat(y,ypred)`: Creates a confusion matrix based on the predictions `ypred` and actual class labels `y`.
- `confusionchart(C)`: Makes a pretty chart from the confusion matrix `C`.
- `perfcurve(y,ytilde,pos)`: Returns performance metrics for binary classifier given class labels `y`, scores `ytilde`, and positive class label `pos`. Default is FPR/TPR for an ROC curve, but other options are available.

Linear Systems

- `A\B`: If `B` has multiple columns, solves the linear system (or least squares problem) for each column individually.

Logistic Regression

- `mnrfit(x,y)`: Fits multinomial logistic regression model to data.
- `mnrval(B,x)`: Evaluates logistic regression model with coefficients `B` on data `x`.

Bibliography

Articles

[15] George Marsaglia. “Random numbers fall mainly in the planes”. In: *Proceedings of the National Academy of Sciences of the United States of America* 61.1 (1968), page 25 (cited on page 111).

Online Sources

[2] *Bomb Sight: Mapping the WWII London Blitz Bomb Census*. URL: http://bombsight.org/about/ (cited on page 223).

[3] Jorge Cham. *PhD Comics 1156*. URL: http://phdcomics.com/comics.php?f=1156 (cited on page 195).

[5] *Dota 2 Wiki: Random Distribution*. URL: https://dota2.fandom.com/wiki/Random_distribution (cited on page 108).

[6] *Encyclopedia Britannica*. URL: https://www.britannica.com/science/Mercator-projection (cited on page 164).

[7] Michael Galarnyk. *Explaining the 68-95-99.7 rule for a Normal Distribution*. URL: https://towardsdatascience.com/understanding-the-68-95-99-7-rule-for-a-normal-distribution-b7b7cbf760c2 (cited on page 82).

[9] Mads Haahr. *RANDOM.ORG*. URL: https://www.random.org/ (cited on pages 101, 108).

[11] Clay Math Institute. *Millennium Problems*. URL: https://www.claymath.org/millennium-problems (cited on page 225).

[13] *LAPACK – Linear Algebra PACKage*. URL: https://www.netlib.org/lapack/ (cited on page 117).

[14] *LAPACK in Matlab*. URL: https://www.mathworks.com/help/matlab/math/lapack-in-matlab.html (cited on page 117).

[16] Rachael Meager. *Untitled Twitter Thread*. URL: https://twitter.com/economeager/status/1395791301627596806 (cited on page 208).

[18] Cleve B Moler. *Cleve's Corner: Cleve Moler on Mathematics and Computing*. URL: https://blogs.mathworks.com/cleve/ (cited on page 42).

[19] ResNeXtGuesser. *Neural Net Guesses Memes*. 2021. URL: https://twitter.com/resnextguesser/status/1455331335904514053 (cited on page 260).

[20] Sonja Surjanovic and Derek Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. URL: https://www.sfu.ca/~ssurjano/index.html (cited on page 92).

[22] Nicholas Trefethen. *New BMI (Body Mass Index)*. URL: https://people.maths.ox.ac.uk/trefethen/bmi.html (cited on page 186).

[24] Tyler Vigen. *Spurious Correlations*. URL: https://www.tylervigen.com/spurious-correlations (cited on page 208).

Technical Reports

[1] Lawrence E Bassham III et al. *Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*. 2010 (cited on page 113).

[12] W Kahan. *Implementation of algorithms. part 1*. Technical report. University of California, Berkeley, Department of Computer Sciences, 1973 (cited on page 45).

Books

[4] James W Demmel. *Applied numerical linear algebra*. SIAM, 1997 (cited on page 117).

[8] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1989 (cited on page 24).

[10] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002 (cited on page 32).

[17] Cleve B Moler. *Numerical computing with MATLAB*. SIAM, 2004 (cited on pages 101, 108, 111).

[21] John Taylor. *Introduction to error analysis, the study of uncertainties in physical measurements*. 1997 (cited on page 31).

[23] Charles F Van Loan and G Golub. *Matrix computations (Johns Hopkins studies in mathematical sciences)*. The Johns Hopkins University Press, 1996 (cited on pages 146, 147).

Index