Numerical Methods

Xuemei Chen

University of San Francisco

(Math 375: Numerical Analysis in Fall 2017)

Contents

Preliminaries

Before this course, you need to be familiar with

- Calculus I
- Calculus II: especially Sequences and Series
- Linear Algebra
  1. Vectors, norm, dot product
  2. Linear Independence and Dependence
  3. Span, subspace, bases, dimension of a subspace/vector space
  4. Properties of orthogonal or orthonormal sets/bases; Gram-Schmidt algorithm
  5. Matrix multiplication; Determinant; Inverse
  6. Solving linear equations: algorithm, existence and uniqueness.
  7. Projections; Least square
  8. Eigenvalues and Eigenvectors
  9. SVD

Notations

- $\{x|$ descriptions of $x\}$ is the set of all x that meets the description after the ":" sign. For example, $\{x|x^2-1<0\}$ is the set of all number x such that $x^2-1<0$. We can solve this inequality further and see that $\{x|x^2-1<0\}=(-1,1)$

  Remark: Some books use colon instead of verticle bar as $\{x:$ descriptions of $x\}$

- $\in$: belongs to/is contained in. For example, $a \in \{x|x^2-1<0\}$ means that a is a number satisfying $a^2-1<0$.
- s.t.: such that

Chapter 1

Solving nonlinear equations

If you recall the equations that you are able to solve, you would realize that there are very few. You know how to solve $x^2 - x - 1 = 0$ using the famous quadratic formula, but have you ever wondered formulas for finding roots of polynomials whose degree is higher than 2? Unfortunately, even as simple as a polynomial equation $x^5 - x - 1 = 0$, a math Ph.D feels hopeless to find an exact answer by hand (so you shouldn't feel bad). In fact, there are no formula for polynomials of degree bigger than 4. Equations like $\cos x = x$? does not have a closed form solution either.

When we need to solve nonlinear equations that frequently occur in every aspect of sciences and applications, we use iterative methods.

1.1 Bisection Method

This is a simple, yet efficient method that can be covered in Calculus I as an application of the Intermediate Value Theorem (IVT).

Theorem 1.1 (Intermediate Value Theorem)**.** *If $f(x)$ is continuous on $[a, b]$, then for any m that is in between $f(a)$ and $f(b)$, there exists a number $c \in [a, b]$ such that $f(c) = m$.*

Take $f(x) = x^5 - x - 1$ as an example, whose graph is shown on the left of Figure 1.1. Since $f(1) = -1 < 0$ and $f(1.5) \approx 5.09 > 0$, then by IVT, there is $c \in [1, 1.5]$ such that $f(c) = 0$ and this c is exactly the root that we are looking for.

To have a more precise estimate of c, we evaluate f at the middle point $\dfrac{1+1.5}{2} = 1.25$. $f(1.25) \approx 0.8 > 0$. By IVT again, we conclude that the root $c \in [1, 1.25]$. We can keep bisecting the interval and eventually find the root to a sufficient precision. See Figure 1.1.

Note that the key of this Bisection method are:

1. The function needs to be continuous.
2. One needs to be given initial points a, b such that $f(a)f(b) < 0$.

1.2 Newton's Method

Newton's method is a little more sophisticated: it involves derivatives, but it is still Calc I material. This is usually how a calculator or a computer finds a root.

If we are to find the root of $y = f(x)$, we start with a random initial guess x_1. Consider the the tangent line to the curve $y = f(x)$, at $P_0 = (x_0, f(x_0))$. The idea behind Newton's method

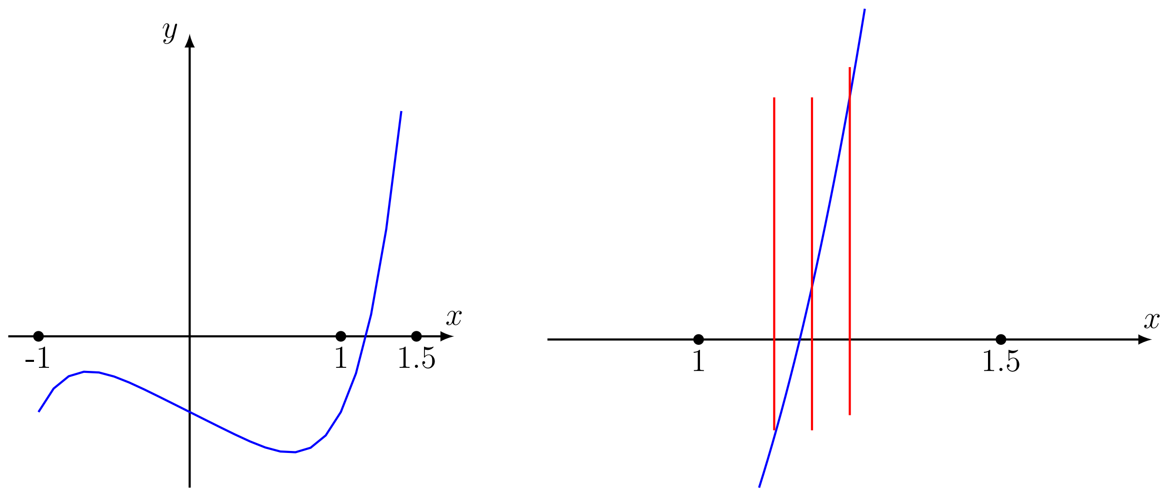


Figure 1.1: Bisection Method

is linearization. Since the tangent line (linearization of f) is close to f, then its x-intercept, x_2, is close to the x-intercept of the curve $y = f(x)$. We can easily find x_2.

The tangent line at the point $(x_0, f(x_0))$ is

$$y - f(x_0) = f'(x_0)(x - x_0).$$

Plugging in $y = 0$ to solve for the x-intercept:

$$0 - f(x_0) = f'(x_0)(x - x_0) \Longrightarrow x = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

So $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$. We use x_1 as the next approximation and keep repeating this process, as shown in Figure 1.2, where we gain this iteration formula

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k \geq 0 \tag{1.1}$$

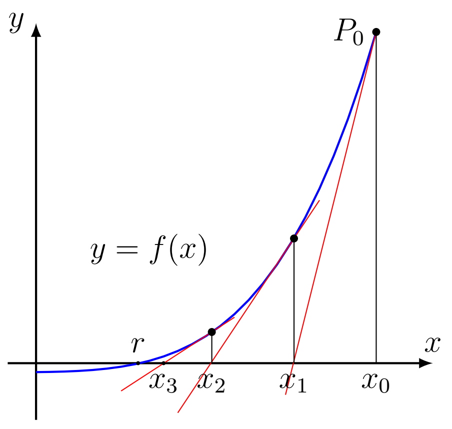


Figure 1.2: Newton's method

Example 1.2. Use Newton's method to approximate $\sqrt{2}$.

Let $f(x) = x^2 - 2$, which makes $\sqrt{2}$ a root of $f(x)$. $f'(x) = 2x$. Let $x_0 = 4$.

$$x_{k+1} = x_k - \frac{x_k^2 - 2}{2x_k} = \frac{x_k}{2} + \frac{1}{x_k}$$

$x_1 = 2 + 1/4 = 9/4$
$x_2 = 9/8 + 4/9 = 113/72$
...

The two drawbacks of the Newton's method are: (1) It requires the knowledge of the derivative function; (2) It may not converge if you have an unlucky initial guess x_0. For example, if one picks $x_1 = 0$ for finding root of $y = x^5 - x - 1$, the sequence will not converge to the root (try to draw tangent lines on your own). However, we do have convergence if the initial guess is close enough.

Theorem 1.3 ([1, Theorem 4.3.1])**.** *If the second derivative of $f(x)$ is continuous, and x_0 is sufficiently close to a root r of f, then Newton's method converges to r and ultimately the convergence rate is quadratic.*

The next section introduces the secant method which is Newton's method without requiring derivatives.

1.3 Secant Method

The secant method is defined by taking $f'(x_n)$ to be

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

So the iteration formula for the secant method is:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}.$$

1.4 Fixed point methods

x is called a fixed point of $\varphi(x)$ if $\varphi(x) = x$.

Example 1.4. Find fixed point of $\varphi(x) = x^2$.

This is to solve $x = x^2 \Longrightarrow x(x-1) = 0 \Longrightarrow x = 1, 0$

What about finding fixed points of $\varphi(x) = e^{-x}$ since we can't solve $e^{-x} = x$ by hand anymore? A common way is to iterate using the formula

$$x_{k+1} = \varphi(x_k), \qquad k \geq 0$$

which in this example will be $x_0 = 0, x_1 = \varphi(x_0) = 1, x_2 = \varphi(x_1) = e^{-1}, ...$

Finding the fixed point of $\varphi(x)$ is equivalent to finding the root of $\varphi(x) - x$, which means we can always use Newton's method to find a fixed point. Using the same example, we let

$f(x) = e^{-x} - x$, then $f'(x) = -e^{-x} - 1$. So the Newton' iteration is $x_{k+1} = x_k + \dfrac{e^{-x_k} - x_k}{e^{-x_k} + 1}$. Still letting $x_0 = 0$, we have $x_1 = 0 + 1/2 = 1/2$, which is already a better approximation than the fixed point iteration.

Theorem 1.5. *Assume $\varphi \in C^1$ and $|\varphi'(x)| < 1$ in some interval $[r - \delta, r + \delta]$ where r is a fixed point of φ. If $x_0 \in [r - \delta, r + \delta]$ and we use iteration $x_{k+1} = \varphi(x_k)$, then $\lim_{k\to\infty} x_k = r$.*

Proof. See page 95 of [1]. $\square$

Newton's method (1.1) can be considered as a fixed point iteration. To be specific, (1.1) is equivalent to $x_{k+1} = \varphi(x_k)$ with $\varphi(x) = x - \dfrac{f(x)}{f'(x)}$. We can apply Theorem 1.5 to pick a good initial point for Newton's method.

Example 1.6. We are trying to use Newton's method to find the root of $f(x) = x^3 - 1$. The Newton's iteration is $x_{k+1} = \varphi(x_k)$, where $\varphi(x) = x - \dfrac{f(x)}{f'(x)} = x - \dfrac{x^3 - 1}{3x^2} = \dfrac{2}{3}x + \dfrac{1}{3x^2}$. $\varphi'(x) = \dfrac{2}{3} - \dfrac{2}{3}x^{-3}$.

$$|\varphi'(x)| < 1 \Longrightarrow \left|\frac{2}{3} - \frac{2}{3}x^{-3}\right| < 1 \Longrightarrow |\frac{1}{x^3} - 1| < \frac{3}{2} \Longrightarrow -\frac{3}{2} < \frac{1}{x^3} - 1 < \frac{3}{2} \Longrightarrow x^3 > \frac{2}{5} \text{ or } x^3 < -2$$

We can pick x_0 from the interval $(\left(\dfrac{2}{5}\right)^{1/3}, 2 - \left(\dfrac{2}{5}\right)^{1/3})$. Notice that the interval has to be centered around 1.

Remark 1.7. Note that Theorem 1.5 is only a sufficient condition. Even if $|\varphi'(x_0)| < 1$, x_0 can be still a fine initial point. For example, any x_0 can work in Example 1.6.

Chapter 1 Exercises

* means extra credit problems

1 Use Newton's method to approximate $\sqrt{2}$. Let $x_0 = 1$. Compute 2 iterates only.

2 Prove that Newton's method will converge to 0 given any initial value x_0 if we are solving $x^2 = 0$.

3 Write down the first three iterates of the secant method for solving $x^2 - 3 = 0$, starting with $x_0 = 0$ and $x_1 = 1$.

4 We can compute 1/3 by solving $f(x) = 0$ with $f(x) = x^{-1} - 3$.

- (a) Write down the Newton iteration for this problem, and compute by hand the first 2 Newton iterates for approximating 1/3, starting with $x_0 = 0.5$.
- (b) What happens if you start with $x_0 = 1$?
- (c) *In the case of (b), show that the iterates $x_k \to -\infty$ as $k \to \infty$.

(d) Use the theory of fixed point iteration to determine an interval about $1/3$ from which Newton's method will converge to $1/3$.

5 Let function $\varphi(x) = (x^2 + 4)/5$.

(a) Find the fixed point(s) of $\varphi(x)$.

(b) Would the fixed point iteration, $x_{k+1} = \varphi(x_k)$, converge to a fixed point in the interval $[0, 2]$ for all initial gueses $x_0 \in [0, 2]$?

(c) *Find a function $f(x)$ such that its Newton iterations are $x_{k+1} = \varphi(x_k)$. (Hint: You need to solve a separable differential equation (Calc II material))

6 Compare Bisection method and Newton's method. List their pros and cons. Think of a way to combine Bisection method and Newton's method to overcome the drawbacks of the Newton's method.

Chapter 2

Floating point arithmetic

Try add up 0.1 and 0.2 in Python, you will get

```
>>> 0.1 + 0.2
0.30000000000000004
```

This can cause serious issues and create bugs. See Section 5.1 of [1]. The above error is caused by rounding in floating point numbers, with which numerical analysis is traditionally concerned.

Floating point numbers are represented in terms of a base β, a precision p, and an exponent e. For example, in base 10, 0.34 can be represented as 3.4×10^{-1} or 34×10^{-2}. In order to guarantee uniqueness, we take a floating point number to have the specific form.

$$\pm (d_0 + d_1\beta^{-1} + \cdots + d_{p-1}\beta^{-(p-1)})\beta^e, \tag{2.1}$$

where each d_i is an integer in $[0, \beta)$ and $d_0 \neq 0$. Such a representation is said to be a *normalized floating point number*. Here are a few examples where $\beta = 10, p = 5$ and e is in the range $-10 \leq e \leq 10$:

$$1.2345 \times 10^8, \qquad 3.4000 \times 10^{-2}, \qquad -5.6780 \times 10^5$$

In these examples, we are using base $\beta = 10$ to ease into the material. Computers work more naturally in binary (base 2). **Please review binary representation**. When $\beta = 2$, each digit d_i is 0 or 1 in equation (2.1). To review, we have

- $11.0101_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 2 + 1 + 0.25 + 0.0625 = 3.3125$
- $1100_2 = 1.1_2 \times 2^3 = (1 \cdot 2^0 + 1 \cdot 2^{-1}) \cdot 2^3 = 12$
- $0.01_2 = 1.00_2 \times 2^{-2} = 0.25$

The number 0.1 has an exact representation in base 10. However, 0.1 does not have a finite binary expansion. In fact

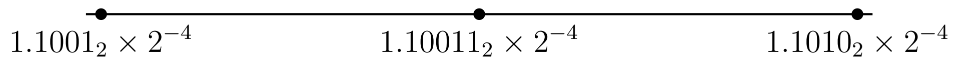
$$0.1 = 0.0\overline{0011}_2$$

meaning $0.1 = 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + \cdots$. The representation has to be rounded at some point, and this is causing the computation inaccuracy at the beginning of this section.

There are usually 4 rounding models: Rounding down, Rounding up, Rounding towards 0, and rounding to nearest. The default IEEE standard is rounding to nearest, that is, either round down or round up, whichever is closer. In case of a tie, it is the one whose least significant (rightmost) bit is 0.

For convenience of illustration, we use examples where we retain 4 digits after the binary point. And we shall use the round to nearest model.

Example 2.1. $0.1 = 1.1001100\cdots_2 \times 2^{-4}$. But we have to round this number so that there are only 4 digits after the binary point. Rounding up will be $u = 1.1010_2 \times 2^{-4}$. Rounding down will be $d = 1.1001_2 \times 2^{-4}$. To see which one is nearest, we can compute the midpoint of u and d.



0.1 is clearly bigger than the middle point, so we pick $1.1010_2 \times 2^{-4}$ with rounding to the nearest.

Example 2.2 (Addition)**.** Suppose we retain 4 digits after the binary point. To add up two numbers with different powers, we first need to adjust one of the numbers to align the significands (use bigger power), as:

$1.1000_2 \times 2^1 \oplus 1.1001_2 \times 2^{-1} = 1.1000_2 \times 2^1 \oplus fl(0.011001_2) \times 2^1 = 1.1000_2 \times 2^1 \oplus 0.0110_2 \times 2^1 = fl(1.1110_2) \times 2^1 = 1.1110_2 \times 2^1$.

Almost all machines today use IEEE-754 floating point arithmetic, and almost all platforms map Python floats to IEEE-754 double precision. This standard has 64 bits, with 1 bit for the sign, 11 for the exponent, and 52 for the significand.

Use the IEEE-754 floating point number to represent,

$$0.1 = +1.\underbrace{1001100110011001100110011001100110011001100110011010_2}_{\text{52 digits}} \times 2^{-4},$$

which is approximately `0.1000000000000000055511151231257827021181583404541015625` in decimal.

Machine precision is related to the number β and p in (2.1). In the case when $\beta = 10, p = 5$. The closest number to 1.2345 is 1.2346 (or 1.2344). Machine precision is exactly that gap 0.0001=10^{-4}. Moreover, this gap is relative. The closest number to 1.2345×10^4 is 1.2346×10^4 (take the bigger one again). This means we can not represent any number between 12345 and 12346 with this machine. The gap is a lot bigger now (=1), but the relative gap is $\frac{1}{10^4}$, still 0.0001.

In general, machine precision $\epsilon_m = \beta^{p-1}$. With the IEEE-754 standard, the machine precision is $\epsilon_{IEEE} = 2^{-52} \approx 2.22 \times 10^{-16}$. Addition (or subtraction) wise, we have $a \oplus a\epsilon = a$ if $\epsilon < \epsilon_{IEEE}$.

2.1 Fundamental Axiom of Floating Point Arithmetic

If we use $fl(x)$ to represent the floating point number of x, then we always obey that $fl(x) = x(1+\epsilon)$ such that $|\epsilon| \le \epsilon_m$, where ϵ_m is the machine precision. This can also be expressed as $\left|\frac{fl(x)-x}{x}\right| \le \epsilon_m$, and interpreted as "relative error is bounded by the machine precision."

Example 2.3. Let us use base 10 with 1 digit precision as an example, in which case the machine precision is $\epsilon_m = 10^{-1}$. If we use the rounding to nearest method,

1.23 becomes 1.2. Relative error=$\left|\frac{1.23-1.2}{1.23}\right| \approx 0.02$

10.61, is first expressed as 1.061*10, then round to 1.1*10. Relative error=$\left|\frac{10.61-11}{10.61}\right| \approx 0.04$

Not only the rounding needs to have small relative error, all the operations $(+, -, \times, \div)$ should be stable in the same way:

Fundamental axiom of floating point arithmetic: Let $*$ be one of the operations $(+, -, \times, \div)$, and let $\circledast$ be its floating point analogue, then for all **floating point numbers** x, y, we have that

$$x \circledast y = (x * y)(1 + \epsilon), \text{ for some } \epsilon \text{ such that } |\epsilon| \leq \epsilon_m.$$

or equivalently

$$\left| \frac{x \circledast y - x * y}{x * y} \right| \leq \epsilon_m.$$

Let us check quickly check this axiom with the operation $+$ on one example (insanity check).

Example 2.4. We assume a machine, base 2, can only have 4 digits precision after the binary point, rounding to the nearest. Let $x = 1.1010_2 \times 2^{-4}$ and $y = 1.1010_2 \times 2^{-3}$ be two floating point numbers. (x is approximately 0.1 by Example 2.1 and y is approximately 0.2 in decimal.)

Similar to Example 2.2, we first rewrite x as $0.11010_2 \times 2^{-3}$, which rounds to $0.1110_2 \times 2^{-3}$ (break the tie by picking the one whose right most bit is 0).

$x \oplus y = fl(0.1110_2 \times 2^{-3} + 1.1010_2 \times 2^{-3}) = fl(10.1000_2 \times 2^{-3}) = 1.0100_2 \times 2^{-2} = 0.3125.$

The relative error for this addition is

$$\left| \frac{x \oplus y - (x + y)}{x + y} \right| = \left| \frac{1.0100_2 \times 2^{-2} - 1.001110_2 \times 2^{-2}}{1.001110_2 \times 2^{-2}} \right| = \frac{0.000010_2}{1.001110_2} \approx 0.0256 < 2^{-4} = \epsilon_m.$$

Example 2.5. If we assume a machine, base 2, can only have 4 digits precision after the binary point. This is how it adds up 0.1 and 0.2.

Step 1: 0.1 is rounded to $1.1010_2 \times 2^{-4} = fl(0.1)$.

Step 2: 0.2 is rounded to $1.1010_2 \times 2^{-3} = fl(0.2)$.

Step 3: $fl(0.1) \oplus fl(0.2) = 0.3125$ as shown in Example 2.4.

The overall relative error is $0.0125/0.3 \approx 0.042$.

To sum up, if we use $m(\cdot)$ to indiate the machine output using floating pont numbers, then $m(x + y) = fl(x) \oplus fl(y) = (x(1 + \epsilon_1) + y(1 + \epsilon_2))(1 + \epsilon_3)$

Chapter 2 Exercises

1. Convert binary to decimal.
   (a) 110_2
   (b) 11.0101_2

2. Directly add and subtract in binary (No rounding involved).
   (a) $0.111_2 + 11.101_2$
   (b) $1.0101_2 - 1.10_2$
   (c) $1.1010_2 \times 2^{-4} + 1.1010_2 \times 2^{-3}$ as in Example 2.4. Convert it to decimal with a calculator.

3. Compute the **normalized** floating point numbers

   (a) 10.3456, base 10, retain 4 digits after point, rounding up.
   (b) 1.0101_2, base 2, retain 2 digits after point, rounding up.
   (c) 1.0101_2, base 2, retain 2 digits after point, rounding down.
   (d) 1.0101_2, base 2, retain 2 digits after point, rounding to the nearest (decide which one is closer by finding the midpoint of answers in (b) and (c)).

4. Compute the machine result of 0.1+0.2 if we retain 2 digits precision after the binary point.

5. *In Python

```
>>>  (0.1 + 0.2) + 0.3
0.6000000000000001
>>>  0.1 + (0.2 + 0.3)
0.6
```

   Explain this phenomenon with a more limited machine where we only retain 2 digits after the binary point.

6. This problem is from Chapter 5 problem 15 of [1].

   In the 1991 Gulf War, the Patriot misslie defense system failed due to roundoff error. The troubles stemmed from a computer that performed the tracking calculations with an internal clock whose integer values in tenths of a second were converted to seconds by multiplying by a 24-bit binary approximation to 0.1:

$$0.1 \approx 0.00011001100110011001100_2$$

   (a) Convert the binary number above to a fraction. Call it x.
   (b) Compute the absolute difference between 0.1 and x.
   (c) What is the time error in seconds after 100 hours of operation (i.e., the value of $|360,000 - 3,600,000x|$)?

   On February 25, 1991, a Patriot battery system, which was to protet the Dhahran Air Base, had been operating for over 100 consecutive hours. The roundoff error caused the system not to track an incoming Scud missle, which slipped through the defense system and detonated on US Army barracks, killing 28 American soldiers.

Chapter 3

Linear Algebra Essentials

3.1 Vector

The letters towards the end of the alphabet, like x, y, z, usually indicate vectors. They are considered as column vectors unless otherwise stated. For a vector x, x_i will denote its ith coordinate. The square brackets $[\cdot]$ is for matrices. Since vectors are a special kind of matrices, we have

- $x = [x_1, x_2 \cdots, x_n]$ is a row vector.
- $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ is a column vector.
- $x = [x_1, x_2 \cdots, x_n]^T$ is a column vector.
- $x = (x_1, x_2 \cdots, x_n)$ is a column vector. Yes, if we use parenthesis, it is a column vector!

In consideration of saving space, you will see the last two expressions often.

Let $x = (x_1, x_2 \cdots, x_n)$ and $y = (y_1, y_2, \cdots, y_n)$ be two real vectors in $\mathbb{R}^n$, then the *inner product* of x and y is defined as

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i.$$

Notice $\langle x, y \rangle = x^T y$ (or $y^T x$).

If we switch order, xy^T will result in a rank-1 matrix, which is called the *outer product* of x and y.

Two vectors x, y are called *orthogonal* if $\langle x, y \rangle = 0$. x is called a unit (norm) vector if $\|x\| = 1$. It is obvious that $\langle x, x \rangle = \|x\|_2^2$. We use the notion $\|\cdot\|_2$ here because in this course, we will learn a more general notion of norm.

For several vectors $\{v_1, v_2, \cdots, v_n\}$, they are called

- *Orthogonal* if $v_i^T v_j = 0, \forall\ i \neq j$.
- *Orthonormal* if $\begin{cases} v_i^T v_j = 0, & \forall\ i \neq j \\ v_i^T v_i = 1, & \forall\ 1 \leq i \leq n \end{cases}$

Definition 3.1. [Vector norm] A *norm* for a vector is a function $\|\cdot\|$ satisfying
(i) $\|v\| \geq 0$ for all vector v, with equality if and only if $v = 0$ (positive definite);
(ii) $\|\alpha v\| = |\alpha|\|v\|$ for any scalar α and any vector v (scalable);
(iii) $\|v + w\| \leq \|v\| + \|w\|$ for all vector v, w (triangle inequality).

The most common norm is probably the class of ℓ_p norms. For any $p \geq 1$, define:

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$

Among the ℓ_p norm class, the frequently used ones are the ℓ_1 norm:

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|,$$

the ∞-norm

$$\|x\|_\infty = \max_i |x_i|,$$

and of course the ℓ_2 norm (Euclidean norm) that we are very familiar with.

The ℓ_2 norm coincides with the length that human begings perceive, but other norms are helpful as well, both in application and theory. For example, the ℓ_1 norm is also called the taxi cab distance because it can be interepreted as the driving distance in a neighborhood where streets are in grid.

Example 3.2. Let $x = (1, 1, -2)$, then $\|x\|_2 = \sqrt{6}$, $\|x\|_1 = 4$, $\|x\|_\infty = 1$.

3.2 Matrix

$A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} r_1^T \\ r_2^T \\ \vdots \\ r_m^T \end{bmatrix} = [c_1, c_2, \cdots, c_n]$ is an $m \times n$ matrix, where r_i^T is the ith row of this matrix and c_j is the jth column of this matrix.

Let $x = [x_1, \cdots, x_n]^T$, then there are three ways to view the multiplication Ax.

(1) $Ax = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{mn}x_n \end{bmatrix}$

(2) $Ax = \begin{bmatrix} r_1^T \\ r_2^T \\ \vdots \\ r_m^T \end{bmatrix} x = \begin{bmatrix} r_1^T x \\ r_2^T x \\ \vdots \\ r_m^T x \end{bmatrix}$.

(3) $Ax = [c_1, c_2, \cdots, c_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} x_i c_i$. Ax is a linear combination of columns of A.

The null space or kernel of A is defined as

$$N(A) = \{x : Ax = 0\}$$

Given $A \in \mathbb{R}^{n\times n}$, if $Ax = \lambda x$ for some $x \neq 0$, then x is an eigenvector of A, and λ is the corresponding eigenvalue. For example, we have $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, which means that $[x, 1]^T$ is an eigenvector of $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$, with the corresponding eigenvalue 3.

To figure out all the eigenvalues and its corresponding eigenvectors of A, we need to first find roots of $\det(A - \lambda I)$, and second find the null space of $A - \lambda I$ for each root.

Example 3.3. Find eigenvalues and eigenvectors of $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$.

$\det \begin{bmatrix} 1-\lambda & 2 \\ 2 & 1-\lambda \end{bmatrix} = (1-\lambda)^2 - 4 = (1-\lambda-2)(1-\lambda+2) = (-1-\lambda)(3-\lambda)$.

$\lambda_1 = -1$, solve $\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} x = 0$, $u_1 = [1, -1]^T$.

$\lambda_2 = 3$, solve $\begin{bmatrix} -2 & 2 \\ 2 & -2 \end{bmatrix} x = 0, u_2 = [1, 1]^T$.

$A \in \mathbb{R}^{n\times n}$ is symmetric if $A = A^T$.

$A \in \mathbb{R}^{n\times n}$ is called diagonalizable if we can find n independent eigenvectors. (This is not the official definition of diagonalizability, but it's an equivalent one.) A symmetric matrix must be diagonalizable. Figure 3.1 shows the ownerships of these matrices.

Block matrix multiplication

Block matrix multiplication is a very useful tool. For example, it will be used effectively later when analyzing QR decomposition and SVD. We first show an example.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 5 & 4 & 0 \\ -2 & 2 & 3 \end{bmatrix} = \left[\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & 0 & -1 \\ 0 & 1 & 0 \end{array}\right] \left[\begin{array}{ccc} 2 & 3 & 4 \\ \hline 5 & 4 & 0 \\ -2 & 2 & 3 \end{array}\right] := \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & A \end{bmatrix} \begin{bmatrix} \rho_1 \\ B \end{bmatrix}$$

where the bold $\mathbf{0}$ are 0 vectors/matrices, and $A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$, $\rho_1 = [2\ 3\ 4], B = \begin{bmatrix} 5 & 4 & 0 \\ -2 & 2 & 3 \end{bmatrix}$.

Block matrix multiplication says that you can treat each block as an entry, so we can continue our computation as

$$\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & A \end{bmatrix} \begin{bmatrix} \rho_1 \\ B \end{bmatrix} = \begin{bmatrix} 1 \cdot \rho_1 + \mathbf{0}B \\ \mathbf{0}\rho_1 + AB \end{bmatrix} = \begin{bmatrix} \rho_1 \\ AB \end{bmatrix}. \tag{3.1}$$

Eventually $\begin{bmatrix} \rho_1 \\ AB \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 \\ -2 & 2 & 3 \\ -5 & -4 & 0 \end{bmatrix}$ as expected.

The equation (3.1) shows that left multiplying a matrix of the form $\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & A \end{bmatrix}$ will always preserve the first row of that matrix.

We summarize the two rules of block matrix multiplication:

(i) After partitioning into blocks, the restructured matrices can be multiplied. In the example above, this is saying that the 2×2 can be multiplied by a 2×1.

(ii) Each corresponding blocks can be multiplied. In the example above, this is saying that the four operations $1 \cdot \rho, \mathbf{0}B, \mathbf{0}\rho_1, AB$ are legit.

For instance, the partitioning $\left[\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & 0 & -1 \\ 0 & 1 & 0 \end{array}\right] \left[\begin{array}{ccc} 2 & 3 & 4 \\ 5 & 4 & 0 \\ \hline -2 & 2 & 3 \end{array}\right]$ satisfies (i) but violates (ii). The partitioning $\left[\begin{array}{c|c|c} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{array}\right] \left[\begin{array}{ccc} 2 & 3 & 4 \\ 5 & 4 & 0 \\ \hline -2 & 2 & 3 \end{array}\right]$ violates (i) already.

3.3 Positive definite matrices

Given a symmetric matrix A, there are 3 equivalent criteria of positive definite matrices.

I All eigenvalues of A are positive.

II For any vector x, we have $x^T Ax \geq 0$ and equality holds only when $x = 0$.

III All leading principal minors are positive.

It is also useful to introduce the concept of positive semidefinite matrices. Given a symmetric matrix A, there are 2 equivalent criteria of positive semidefinite matrix.

I All eigenvalues of A are nonnegative.

II For any vector x, we have $x^T Ax \geq 0$.

Note: All leading pricipal minors are nonnegative is not a characterization of positive semidefinite matrix.

Example 3.4. Verify that $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ is positive definite using all three criteria.

I. $\det \begin{bmatrix} 1-\lambda & 0 & 0 \\ 0 & 2-\lambda & 1 \\ 0 & 1 & 2-\lambda \end{bmatrix} = (1-\lambda)[(2-\lambda)^2 - 1] = (1-\lambda)(2-\lambda+1)(2-\lambda-1)$
eigenvalues are 1,3, all positive.
II. $x^T Ax = x_1^2 + 2x_2^2 + 2x_3^2 + 2x_2x_3 = x_1^2 + x_2^2 + x_3^2 + (x_2 + x_3)^2 \geq 0$.
If equality holds, we obviously get $x = 0$.
III. $\det[1] = 1 > 0$, $\det \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} = 2 > 0$, $\det A = 1(4-1) = 3 > 0$.

Theorem 3.5. *If B is invertible, then BB^T and B^TB are both positive definite.*

Proof. We will only prove BB^T is positive definite. The other one is similar and left as an exercise.

First $(BB^T)^T = BB^T$ so BB^T is symmetric.

Second we will use criteria II. For any vector x, $x^T BB^T x = (B^T x)^T B^T x = \|B^T x\|_2^2 \geq 0$. When equality holds, $B^T x = 0$ which implies $x = 0$ because B^T is invertible. $\square$

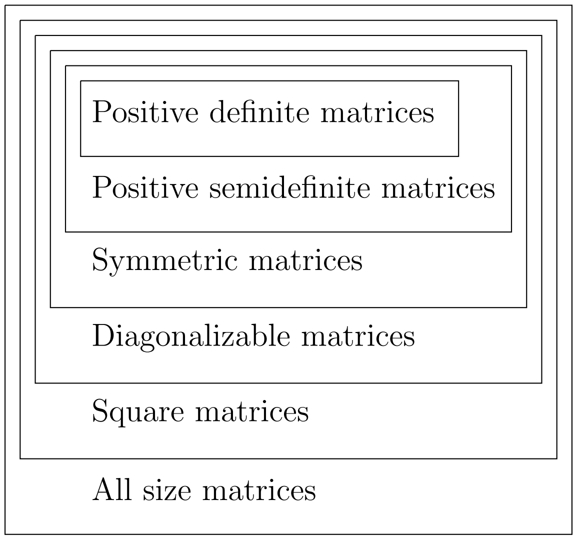


Figure 3.1: Matrices

3.4 Basis

Basis is a very important concept for a vector space. $\{b_1, b_2, \cdots, b_K\}$ is a basis of the vector space V if $\text{span}\{b_1, \cdots, b_K\} = V$ and $\{b_1, b_2, \cdots, b_K\}$ are linearly independent. The vectors in a basis can be considered the building blocks of all vectors in a vector space. For example, using the columns of the identity matrix, $\left\{ e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, e_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots, e_K = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\}$, which is called the *canonical basis of* $\mathbb{R}^K$, we are able to generate all vectors in $\mathbb{R}^K$:

$$\text{For any } x = (x_1, \cdots, x_K) \in \mathbb{R}^K, \quad x = x_1 e_1 + \cdots x_K e_K.$$

Moreover, this "generation" (i.e. the coefficients in front of e_i) is unique due to independence. So another definition of basis is: $\{b_1, b_2, \cdots, b_K\}$ is a basis of the vector space V if every vector in V can be uniquely written as a linear combination of $\{b_1, b_2, \cdots, b_K\}$. Moreover,

$$\dim V = \text{number of vectors in a basis of } V$$

Example 3.6. $\{(1, 2, 0), (1, 1, 0)\}$ is a basis of the xy plane in $\mathbb{R}^3$.

Example 3.7. $\{(1, 2), (1, 1)\}$ is a basis of $\mathbb{R}^2$. $\{\frac{1}{\sqrt{2}}(1, 1), \frac{1}{\sqrt{2}}(-1, 1)\}$ is another basis of $\mathbb{R}^2$.

Notation: Sometimes we will also put the basis vectors as columns of a matrix B and call B is a basis (of some vector space). For example, the identity matrix $I = [e_1, \cdots, e_K]$ is the canonical basis of $\mathbb{R}^K$. $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$ is a basis of $\mathbb{R}^2$ as claimed in Example 3.7.

Remark 3.8. $B' = \{b_1, \cdots, b_K\}$ and $B = [b_1, \cdots, b_K]$ can mean the same basis, so we will abuse the notation and equate B' and B when convenient. But make sure you understand that there is a difference between [and {. { indicates that it is a set, so B' means a set of K vectors. [is for matrix so B is a matrix whose columns are b_i.

There are infinitely many bases of a vector space, and some are better at representing vectors than others. In Example 3.7, the second basis $U_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$ is a better basis because it is an *orthonormal basis.*

Definition 3.9. $U = \{u_1, \cdots, u_K\}$ is an orthonormal basis (ONB) of V if it is a basis of V and the vectors are orthonormal.

U is an ONB if and only if $U^T U = I_K$: the $K \times K$ identity matrix. This is a good way to check whether a matrix is an ONB. (Check U_2 is an ONB for yourself.)

Given a basis $B = \{b_1, \cdots, b_K\}$, we know x can be written as a linear combination of these basis vectors: $x = \sum_{i=1}^{K} c_i b_i$. In order to figure out the coefficients c_i, we need to solve a linear system (solve c from $Bc = x$). This can be annoying. With an ONB, it is a lot better and we have the following useful theorem:

Theorem 3.10. *If $U = \{u_1, \cdots, u_K\}$ is an ONB, and $x = \sum_{i=1}^{K} c_i u_i$, then*

(a) $c_i = u_i^T x = \langle u_i, x\rangle, \forall\ i = 1, \cdots, K$

(b) $\|x\|_2^2 = \sum_i^K |c_i|^2 = \sum_i^K |\langle u_i, x\rangle|^2$.

(c) If U also denotes the matrix whose columns are u_i, then $U^T U = I_K$.

Proof. (a) $u_i^T x = u_i^T(\sum_{j=1}^{K} c_j u_j) = \boxed{\sum_{j=1}^{K} c_j u_i^T u_j} = c_i u_i^T u_i = c_i$. In the boxed summation, if $j \neq i$, the term is 0, so in the end, the only term left is when $j = i$.

(b) $\|x\|_2^2 = \langle x, x\rangle = \langle \sum_{i=1}^{K} c_i u_i, \sum_{j=1}^{K} c_j u_j\rangle = \sum_{i,j} c_i c_j \langle u_i, u_j\rangle = \sum_{i=j} c_i^2 = \sum_{i=1}^{K} c_i^2$

(c) This can be computed directly as

$$U^T U = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_K^T \end{bmatrix} [u_1, u_2, \cdots, u_K] = \begin{bmatrix} u_1^T u_1 & u_1^T u_2 & \cdots & u_1^T u_K \\ u_2^T u_1 & u_2^T u_2 & \cdots & u_2^T u_K \\ \vdots & \vdots & \vdots & \vdots \\ u_K^T u_1 & u_K^T u_2 & \cdots & u_K^T u_K \end{bmatrix} = I_K. \qquad \square$$

A direct implication of Theorem 3.10 (b) is that

$$\|Uc\|_2 = \|c\|_2, \quad \text{for any vector } c \in \mathbb{R}^K. \tag{3.2}$$

because $x = Uc$ if $c = (c_1, \cdots, c_K)$.

Note that UU^T is not necessarily an identity matrix because U could be a tall thin matrix. In fact UU^T is a projection matrix onto span$\{u_1, \cdots, u_K\}$, see towards the end of Section 4.4.

Remark 3.11. If U is an $n \times n$ square matrix whose columns are orthonormal, then U is called an orthonormal matrix (also called orthogonal matrix, unitary matrix). In this case we do have both $UU^T = U^T U = I_n$, and of course (3.2) as well.

3.5 Matrix norm

For a vector norm $\|\cdot\|$, we define the corresponding matrix norm as

$$\|A\| := \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}.$$

Remark: for our class, you can think of sup as max, although they are different.

The matrix norm satisfies the following properties

(i) $\|A\| \geq 0$

(ii) $\|\alpha A\| = |\alpha|\|A\|$

(iii) $\|A+B\| \leq \|A\| + \|B\|$

(iv) $\|AB\| \leq \|A\|\|B\|$

(v) $\|A\| = \sup\limits_{\|x\|=1} \|Ax\|$

(vi) $\|Ax\| \leq \|A\|\|x\|$ (by definition)

Notice the first three properties are the same as in Definition 3.1 because it is also a 'norm'. Properties (iv) and (v) will be most related to the stability of linear systems.

To compute $\|A\|$, the definition tells us to find the biggest ratio $\dfrac{\|Ax\|}{\|x\|}$ among all nonzero vectors. Due to the linearality of A ($A(\alpha x) = \alpha Ax$), it suffices to consider all the vectors with norm 1.

Theorem 3.12. $\|A\| = \sup\limits_{\|x\|=1} \|Ax\|$

We will mostly use 2-norm for matrix norm:

$$\|A\|_2 := \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sup_{\|x\|_2=1} \|Ax\|_2.$$

This matrix 2-norm is also called the operator norm or the spectral norm.

For the identity matrix I, we have

$$\|I\|_2 = \sup_{x\neq 0} \frac{\|Ix\|_2}{\|x\|_2} = \sup_{x\neq 0} \frac{\|x\|_2}{\|x\|_2} = \sup_{x\neq 0} 1 = 1.$$

For the matrix in Example 3.3, $B = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$,

$$\|B\|_2^2 := \sup_{x\neq 0} \frac{\|Bx\|_2^2}{\|x\|_2^2} = \sup_{x_1^2+x_2^2\neq 0} \frac{(x_1+2x_2)^2 + (2x_1+x_2)^2}{x_1^2+x_2^2} \overset{x_1=1,x_2=0}{\geq} 5.$$

So $\|B\|_2 \geq \sqrt{5}$. To compute the exact value of $\|B\|_2$, it is not possible to exhaust all possible x since there are infinitely many. The following theorem tells us how to compute the operator norm.

Theorem 3.13. *For any size matrix A, $\|A\|_2$ is the biggest singular value of A.*

Proof. We first prove this theorem if A is symmetric. Then A has an orthornormal eigenbasis $\{u_i\}_{i=1}^n$ with eigenvalues λ_i. that is, $Au_i = \lambda_i u_i$. We also arrange the eigenvalues decreasingly according to their absolute value, i.e. $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$.

Given a unit norm vector x, we have $x = \sum_{i=1}^n c_i u_i$, and $\sum_{i=1}^n c_i^2 = 1$ by Theorem 3.10.

$$Ax = A\sum_{i=1}^n c_i u_i = \sum_{i=1}^n c_i A u_i = \sum_{i=1}^n c_i \lambda_i u_i.$$

We have $\|Ax\|_2^2 = \sum_{i=1}^n c_i^2 \lambda_i^2$ again by Theorem 3.10.

$$\|A\|_2^2 = \sup_{\|x\|_2=1} \|Ax\|_2^2 = \sup_{\sum c_i^2 = 1} \sum_{i=1}^n c_i^2 \lambda_i^2 = \lambda_1^2.$$

For the last equality, we are considering the problem of how to maximize the weighted sum $\sum_{i=1}^n c_i^2 \lambda_i^2$ given the budget that the weights c_i^2 sum to 1. Of course the maximum is obtained when we put all the weights on the biggest number λ_1^2.

So $\|A\|_2 = |\lambda_1|$, which is the biggest singular values of A. Notice that for a symmetric matrix, the singular values are just the absolute value of the eigenvalues. ☐

Example 3.14. As shown in Example 3.3, singular values of $B = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ are 1 and 3. So $\|B\|_2 = 3$.

Example 3.15. $\left\| \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -2 & 2 \end{bmatrix} \right\|_2 = \sqrt{8}$ as shown in Example 5.5.

Chapter 3 Exercises

1. Given $x = (1, -2, 3), y = (-2, 0, 2)$, find

   (a) Find $x^T y, \|x\|_2, \|x\|_1, \|x\|_\infty$

   (b) Find a vector whose Euclidean norm is 1, and is parallel to y.

   (c) Find a vector that is orthogonal to both x and y. This is the same as finding the null space of what matrix?

2. Find a 3 by 3 orthonormal matrix, then find its inverse. (Don't pick the canonical basis.)

3. If $Au = ru$, simplify $(3A^3 - 4A + 2I)u$ so that A disappears.

4. $B = \begin{bmatrix} 1 & 4 \\ 4 & 7 \end{bmatrix}$.

   (a) Find the eigenvalues and corresponding eigenvectors of B.

   (b) Is this matrix positive semidefinite?

   (c) Find its operator norm.

5. Find the eigenvalues and corresponding eigenvectors of $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 3 \\ 1 & 0 & 2 \end{bmatrix}$. Determine if it is diagonalizable.

6. Let u_1, u_2, u_3 be the 3 columns of the orthonormal matrix you found in Exercise 2, and let $x = (1, 2, 3)$. Find the coefficients c_1, c_2, c_3 in $x = c_1u_1 + c_2u_2 + c_3u_3$.

7. If A is a 2 by 2 matrix that rotates a vector by $\pi/5$ counterclockwise, show that $\|A\|_2 = 1$ directly from its definition.

8. Find the operator norm of $\begin{bmatrix} 1/\sqrt{2} & \sqrt{2} \\ 1/\sqrt{2} & \sqrt{2} \\ 0 & 0 \end{bmatrix}$.

9. Compute x^TAx, where $x = (x_1, x_2, x_3)$, and $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$ (A is symmetric.)

10. Prove that if B is invertible, then B^TB is positive definite.

11. Check that $C = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 3 \end{bmatrix}$ is positive definite using both criteria I and III.

12. Check that $D = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$ is not positive definite using criteria II.

13. Let A be a 3 by 3 matrix, and $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}$.

    (a) How are the rows or columns changed from A to ΣA?

    (b) How are the rows or columns changed from A to $A\Sigma$?

    (c) Use block matrix multiplication to explain either (a) or (b).

Chapter 4

Solving system of linear equations: Direct methods

Solving a linear system $Ax = b$ is everywhere in many applications. There are generally two approaches: Direct methods and iterative (indirect) methods. Direct methods give a direct answer at the end of the algorithm, whereas iterative methods follow an iterative formula where the iterates are approaching the real solution. We talk about direct methods like Gaussian elimination in this chapter and iterative methods in later chapter.

We also care about how efficient an algorithm is. This is to count how many floating point operations (flop) are needed in a computer. Each flop is one of the following operations: $+, -, \times, \div$. For example, the inner product $x^T y$ requires n multiplications and $n-1$ summations, given both vectors have n coordinates. So $x^T y$ needs $2n-1$ flops.

- Matrix-Vector multiplication: Given $A \in \mathbb{R}^{m\times n}, x \in \mathbb{R}^n$, Ax is computing m inner products, so needs $m(2n-1)$ flops.
- Matrix-Matrix multiplication: Given $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$, AB is computing p matrix-vector multiplications. So the calculation of AB requires $pm(2n-1)$ flops. In particular, if both A and B are square matrices, then AB needs $2n^3 - n^2 \approx 2n^3$ flops.

4.1 Gaussian Elimination and LU factorization

Gaussian elimination is a core material in linear algebra and is the procedure one performs when finding key features like linear dependence, rank, null space, basis, dimension, etc. Gauss elimination is the simplest way to solve linear systems of equations by hand, and also the standard method for solving them on computers.

Gaussian elimination is suitable for systems of all sizes, but **in this section our examples and analysis will focus on $Ax = b$ where A is invertible.** When solving $Ax = b$, the Gaussian elimination consists of two procedures: forward elimination and backward substitution. We will review both procedures and do an operation count as well. The following two formula will be useful:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

$\boxed{\text{Forward elimination of Gaussian elimination:}}$ This is to get to an upper triangular matrix with row operations.

One first uses a_{11} to eliminate all entries below

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \longrightarrow \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a'_{n2} & \cdots & a'_{nn} & b'_n \end{array}\right] \tag{4.1}$$

At each row, we need n multiplications and n additions. There are $n-1$ rows to perform, so a total of $(n-1)\cdot 2n = 2n(n-1)$ flops are needed to get to right hand side of (4.1).

Now we uses a'_{22} to eliminate all entries below. This is the same work except we have an $(n-1)\times(n-1)$ system now, so we need $2(n-1)(n-1-1)$ flops.

We keep doing this until the coefficient matrix becomes upper triangular, so there are a total of $2n(n-1)+2(n-1)(n-2)+\cdots+0 = \sum_{i=1}^{n} 2i(i-1) = 2\sum_{i=1}^{n}(i^2-i) = \dfrac{n(n+1)(2n+1)}{3} - n(n+1) \approx \dfrac{2}{3}n^3$ flops.

$\boxed{\text{Backward substitution of Gaussian Elimination:}}$ For backward sub, there are equation form or the matrix form, both of which are illustrated in Example 4.1. We will use the matrix form in our operation counts here.

As the name suggests, we start from the last column and work our way up. One first divides last row by a_{nn}, and then eliminate all entries above a_{nn}.

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{n-1,n} & b_{n-1} \\ 0 & 0 & \cdots & a_{nn} & b_n \end{array}\right] \longrightarrow \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & 0 & b'_1 \\ 0 & a_{22} & \cdots & 0 & b'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & b'_{n-1} \\ 0 & 0 & \cdots & 1 & b'_n \end{array}\right] \tag{4.2}$$

This requires $1 + 2(n-1) = 2n-1$ flops.

For Column $n-1$: $2(n-1)-1$

...

For Column 1: $2-1$

Add up all counts, we have

$$\sum_{i=1}^{n}[2i-1] = n(n+1) - n = n^2$$

Gaussian elimination operation counts:

$$\frac{n(n+1)(2n+1)}{3} - n(n+1) + n^2 \approx \frac{2}{3}n^3$$

Example 4.1. Solve $\left[\begin{array}{ccc|c} 1 & 2 & 2 & 1 \\ 2 & 7 & 7 & 5 \\ 2 & 7 & 9 & 5 \end{array}\right]$.

Forward Elimination:

$$\left[\begin{array}{ccccc}1&2&2&\vdots&1\\2&7&7&\vdots&5\\2&7&9&\vdots&5\end{array}\right]\xrightarrow[\rho1(-2)+\rho3]{\rho1(-2)+\rho_2}\left[\begin{array}{ccccc}1&2&2&\vdots&1\\0&3&3&\vdots&3\\0&3&5&\vdots&3\end{array}\right]\xrightarrow{\rho2(-1)+\rho3}\left[\begin{array}{ccccc}1&2&2&\vdots&1\\0&3&3&\vdots&3\\0&0&2&\vdots&0\end{array}\right].$$

Backward substitution:
(1) Equation form:

$$2x_3 = 0 \Rightarrow x_3 = 0$$
$$3x_2 + 3 * 0 = 3 \Rightarrow x_2 = 1$$
$$x_1 + 2 * 1 + 2 * 0 = 1 \Rightarrow x_1 = -1$$

(2) Matrix form:

$$\left[\begin{array}{ccccc}1&2&2&\vdots&1\\0&3&3&\vdots&3\\0&0&2&\vdots&0\end{array}\right]\xrightarrow{\rho3/2}\left[\begin{array}{ccccc}1&2&2&\vdots&1\\0&3&3&\vdots&3\\0&0&1&\vdots&0\end{array}\right]\xrightarrow[\rho3(-2)+\rho1]{\rho3(-3)+\rho2}\left[\begin{array}{ccccc}1&2&0&\vdots&1\\0&3&0&\vdots&3\\0&0&1&\vdots&0\end{array}\right]$$
$$\xrightarrow{\rho2/(3)}\left[\begin{array}{ccccc}1&2&0&\vdots&1\\0&1&0&\vdots&1\\0&0&1&\vdots&0\end{array}\right]\xrightarrow{\rho2(-2)+\rho1}\left[\begin{array}{ccccc}1&0&0&\vdots&-1\\0&1&0&\vdots&1\\0&0&1&\vdots&0\end{array}\right].$$

Forward elimination=LU factorization

Each row operation corresponds to a matrix (often called elementary row operation matrix). In the forward elimination process of Example 4.1, let $A = \begin{bmatrix}1&2&2\\2&7&7\\2&7&9\end{bmatrix}$, then $L_1A = \begin{bmatrix}1&2&2\\0&3&3\\0&3&5\end{bmatrix}$, where L_1 represents the first two row operations. To figure out L_1, we perform these two row operations on the identity matrix: $\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\xrightarrow[\rho1(-2)+\rho3]{\rho1(-2)+\rho_2}\begin{bmatrix}1&0&0\\-2&1&0\\-2&0&1\end{bmatrix} = L_1$. We can figure out $L_2 = \begin{bmatrix}1&0&0\\0&1&0\\0&-1&1\end{bmatrix}$ the same way, and

$$L_2L_1A = U,$$

where $U = \begin{bmatrix}1&2&2\\0&3&3\\0&0&2\end{bmatrix}$.

So

$$A = (L_2L_1)^{-1}U = L_1^{-2}L_2^{-1}U = LU.$$

L_1, L_2 are very special matrices, and their inverse, and L can be figured out easily:

$$L_1^{-1} = \begin{bmatrix}1&0&0\\2&1&0\\2&0&1\end{bmatrix}, L_2^{-1} = \begin{bmatrix}1&0&0\\0&1&0\\0&1&1\end{bmatrix}, L = L_1^{-1}L_2^{-1} = \begin{bmatrix}1&0&0\\2&1&0\\2&1&1\end{bmatrix}.$$

So we can decompose/factor A as

$$A = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 7 & 7 \\ 2 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 0 & 2 \end{bmatrix} = LU.$$

In general, if there is no row exchange involved in forward elimination, then A can always be written as the product of a lower triangular matrix and an upper triangular matrix. This is called the LU factorization of A. It requires $\frac{2}{3}n^3$ flops as well because it is done through forward elimination. In fact, the operation counts is slightly less because we don't need to consider the right hand side b, but it is still on the order of $\frac{2}{3}n^3$.

A second way to solve $Ax = b$

If the LU factorization of A is already done, then it is very easy to solve $LUx = b$. We set $Ux = y$. We first solve y from $Ly = b$, which is very straightforward as we will see in Example 4.2. Then we solve x from $Ux = y$. We will use the same example to illustrate.

Example 4.2. Suppose we are given $\begin{bmatrix} 1 & 2 & 2 \\ 2 & 7 & 7 \\ 2 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 0 & 2 \end{bmatrix}$. In order to solve $\begin{bmatrix} 1 & 2 & 2 \\ 2 & 7 & 7 \\ 2 & 7 & 9 \end{bmatrix} x = \begin{bmatrix} 1 \\ 5 \\ 5 \end{bmatrix}$,

(1) We first solve $\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} y = \begin{bmatrix} 1 \\ 5 \\ 5 \end{bmatrix}$ $(Ly = b)$.

$$\begin{aligned} y_1 &= 1 \\ 2 * 1 + y_2 = 5 \Rightarrow y_2 &= 3 \\ 2 * 1 + 1 * 3 + y_3 = 5 \Rightarrow y_3 &= 0 \end{aligned}$$

(2) Then we solve $Ux = y$, which is already done in the backward substitution of Example 4.1. It is not a coincidence that the y we solved in (1) is the same as the right hand side of the system after forward elimination.

It is obvious that solving $Ly = b$ and $Ux = y$ cost the same number of operations because the only difference is that $Ly = b$ is solved forwardly, and $Ux = y$ is solved backwardly.

Solve $Ax = b$ via LU decomposition

Step 1:	$A = LU$. (Now equation becomes $LUx = b$.)	$\frac{2}{3}n^3$ flops
Step 2:	Solve $Ly = b$. (y will equal to Ux.)	n^2 flops
Step 3:	Solve $Ux = y$.	n^2 flops

LU decomposition approach is better when we need to solve multiple systems with the same coefficient matrix A.

We compare the flop counts of these two approaches for solving m systems $Ax = b_1, Ax = b_2, \cdots, Ax = b_m$.

Regular Gauss Elimination	$\frac{2}{3}n^3m$ flops
LU approach	$\frac{2}{3}n^3 + 2mn^2$ flops

If $m = n$, then regular GE approach takes around n^4 operations, whereas LU approach operation count is still on the order of n^3. That can be a huge difference when the system is large (i.e. big n).

To compute the inverse of A. It is equivalent to solve the matrix X from $AX = I$. Let the columns of X be $v_1, \cdots, v_n$ and columns of I be $e_1, \cdots, e_n$. Then we are solving n systems $Av_i = e_i, i = 1, 2, \cdots, n$. This requires about $2n^3$ operations according to the discussion above. We summerize the operation counts, using the best direct approach. Given $A, B \in \mathbb{R}^{n\times n}, x, y, b \in \mathbb{R}^{n\times 1}$, and A invertible.

Flop count for various matrix operations

x^Ty	$2n - 1 = O(n)$
Ax	$n(2n-1) = O(n^2)$
AB	$n^2(2n-1) = O(n^3)$
Gauss Jordan – Forward	$\frac{2}{3}n^3 = O(n^3)$
Gauss Jordan – Backward	n^2
$A = LU$	$\frac{2}{3}n^3 = O(n^3)$
Solve $Ax = b$.	$\frac{2}{3}n^3 = O(n^3)$
Find A^{-1}	$2n^3 = O(n^3)$
Solve $AX = B$	$2n^3 = O(n^3)$

We want to point out that there is usually no good reason for ever computing the inverse of a matrix. It does at times happen in certain problems that the entries of A^{-1} have some special physical significance. But whenever A^{-1} is needed merely to calculate $A^{-1}b$ (as in solving $Ax = b$) or a matrix product $A^{-1}B$, A^{-1} should never be calculated explicitly. $A^{-1}b$ is to solve $Ax = b$, and $A^{-1}B$ is to solve $AX = B$ where X is an $n \times n$ matrix. See the chart below, where $A, B \in \mathbb{R}^{n\times n}$, A is invertible, and b is a vector of n coordinates.

	Without Computing A^{-1}	Compute A^{-1} first
Compute $A^{-1}b$	$\frac{2}{3}n^3$	$2n^3 + 2n^2$
Compute $A^{-1}B$	$2n^3$	$2n^3 + 2n^3$

4.2 Pivoting

LU decomposition is not the full story because it does not consider row exchanges, which is necessary in many scenarios.

Consider computing the LU factorization of $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 7 & 9 \end{bmatrix}$. We run into problem right away because $a_{11} = 0$ is not able to eliminate entries below. In this case, a permutation of rows are necessary, and we can, for instance, switch row 1 and row 2. Switching row 1 and row 2 is the same as being left multiplied by the permutation matrix $P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, i.e. $PA = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 2 & 7 & 9 \end{bmatrix}$. Then we can do the same old LU factorization to PA

In general, one can always permute rows of A in such a way that we don't need to do any more row switchings during the forward elimination process. This is the PLU factorization.

$$PA = LU.$$

Example 4.3. Let us look at another system $\left[\begin{array}{cc:c} 10^{-20} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right]$. The solution should be very close to $x_1 = x_2 = 1$, and we are able to get this solution if the computer had infinite precision. Using our Gaussian elimination code with double-precision arithmetic, we get

$$\left[\begin{array}{cc:c} 10^{-20} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right] \rightarrow \left[\begin{array}{cc:c} 10^{-20} & 1 & 1 \\ 0 & -10^{20}+1 & -10^{20}+2 \end{array}\right] \xrightarrow{\rho 2/(1-10^{20})} \left[\begin{array}{cc:c} 10^{-20} & 1 & 1 \\ 0 & 1 & \boxed{1} \end{array}\right]$$

The number 1 in the box is the rounding result of $\dfrac{2-10^{20}}{1-10^{20}}$ (try it for yourself in Python). Another way to think about it is $1 - 10^{20}$ is regarded the same as -10^{20} because the relative error of $1 - 10^{20}$ and 1 is less than machine epsilon. Same with $2 - 10^{20}$.

From there, we solve $x_2 = 1$, $10^{-20}x_1 + 1 = 1 \implies x_1 = 0$, which is far from the actual solution.

A remedy of this is to exchange these two rows:

$$\left[\begin{array}{cc:c} 1 & 1 & 2 \\ 10^{-20} & 1 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cc:c} 1 & 1 & 2 \\ 0 & -10^{-20}+1 & -2\times 10^{-20}+1 \end{array}\right] \xrightarrow{rounding} \left[\begin{array}{cc:c} 1 & 1 & 2 \\ 0 & \boxed{1} & \boxed{1} \end{array}\right]$$

And we can get a much more accurate solution $x_1 = x_2 = 1$.

Partial pivoting

Partial pivoting is a straightforward method of row exchanging in the forward process of Gaussian elimination. At stage k (column k), we will look for the biggest entry in absolute value of that column, and use that entry as a pivot. In Example 4.3, at stage 1, we compare all entries in column 1, and 1 is bigger than 10^{-20}, so we exchange row 1 and row 2 to use 1 as the pivot.

But Partial pivoting is still not perfect. In the system $\left[\begin{array}{cc:c} 10^{-20} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right]$, if we multiply first row by 2×10^{20}, we get $\left[\begin{array}{cc:c} 2 & 2\times 10^{20} & 2\times 10^{20} \\ 1 & 1 & 2 \end{array}\right]$ whose solution is the same as the previous one. According to the partial pivoting strategy, we do not need to exchange rows because 2 is bigger than 1. We leave it as an exercise that no row exchange will still lead to the same catastrophy solution. The real reason that Gaussian elimination (no row exchange) fails for this system is that in row 1, one entry (10^{-20}) is way smaller than the other (1). A much more STABLE solution is to **choose the row whose entries are closest to each other**. This is called the scaled partial pivoting.

Scaled Partial Pivoting

It is very difficult (if not impossible) to ascertain for a general linear system how various pivoting strategies affect the accuracy of the computed solution. One notable and important exception to this statement are systems with positive definite coefficient matrix, for which we will illustrate more in the next section.

For a general linear system, a currently accepted strategy for forward elimination is *Scaled Partial Pivoting.* In this stragety, one calculates initially the "size" s_i of each row i of A

$$s_i = \|\text{row } i\|_\infty = \max_{1\leq j\leq n} |a_{ij}|.$$

Then, at the beginning of step k of the elimination algorithm, suppose we have matrix $A' = (a'_{ij})$, one picks the row that has the biggest $r_{ik} = \dfrac{|a'_{ik}|}{s_i}$ among all the row i that are being consiered. See the following example.

Example 4.4. Use scaled partial pivoting to solve $\begin{bmatrix} 2 & -1 & 7 & 3 & \vdots & 19 \\ 4 & 4 & 0 & 7 & \vdots & 11 \\ 2 & 1 & 3 & 1 & \vdots & 9 \\ 6 & 5 & 4 & -17 & \vdots & -3 \end{bmatrix}$.

First we compute $s_1 = 7, s_2 = 7, s_3 = 3, s_4 = 17.$

Stage 1 (column 1): We compare the four values $\frac{2}{7}, \frac{4}{7}, \frac{2}{3}, \frac{6}{17}$. The biggest is $\frac{2}{3}$ which is coming from $\boxed{\text{row 3}}$. so $a_{31} = 2$ is the pivot and we use it to eliminate other 3 entries in column 1.

$$\begin{bmatrix} 0 & -2 & 4 & 2 & \vdots & 10 \\ 0 & 2 & -6 & 5 & \vdots & -7 \\ 2 & 1 & 3 & 1 & \vdots & 9 \\ 0 & 2 & -5 & -20 & \vdots & -30 \end{bmatrix}$$

Note that by the definition of s_i, all four ratios must be less than or equal to 1, so picking the biggest ratio is picking the pivot that is closes to the biggest entry of the pivot row. At each stage, we can perform a row exchange or not. We choose not in this example.

Stage 2 (column 2): Row 3 is out. We look at entries of column 2, and compare the three values $\frac{2}{7}$(row 1), $\frac{2}{7}$(row 2), $\frac{2}{17}$(row 4). There is a tie, we pick $\boxed{\text{row 1}}$ randomly, and eliminate other 2 entries.

$$\begin{bmatrix} 0 & -2 & 4 & 2 & \vdots & 10 \\ 0 & 0 & -2 & 7 & \vdots & 3 \\ 2 & 1 & 3 & 1 & \vdots & 9 \\ 0 & 0 & -1 & -18 & \vdots & -20 \end{bmatrix}$$

Stage 3 (column 3): Row 1,3 are out. We look at entries of column 3, and compare $\frac{2}{7}$(row 2), $\frac{1}{17}$(row 4). We pick $\boxed{\text{row 2}}$.

$$\begin{bmatrix} 0 & -2 & 4 & 2 & \vdots & 10 \\ 0 & 0 & -2 & 7 & \vdots & 3 \\ 2 & 1 & 3 & 1 & \vdots & 9 \\ 0 & 0 & 0 & -43/2 & \vdots & -43/2 \end{bmatrix} \tag{4.3}$$

After the scaled partial pivoting, we get a matrix who is a permutation of an upper triangular system. We can still use backward substitution:

$$\begin{aligned} &\text{row 4} \quad -\frac{43}{2}x_4 = \frac{43}{2} && \Rightarrow x_4 = 1 \\ &\text{row 2} \quad -2x_3 + 7 = 3 && \Rightarrow x_3 = 2 \\ &\text{row 1} \quad -2x_2 + 8 + 2 = 10 && \Rightarrow x_2 = 0 \\ &\text{row 3} \quad 2x_1 + 0 + 6 + 1 = 9 && \Rightarrow x_1 = 1 \end{aligned}$$

4.3 Cholesky Factorization

For a positive definite matrix A, we do not need to exchange rows. Moreover, its LU decomposition can be made in such a way that L is the transpose of U.

Definition 4.5. For a positive definite matrix A, we can decompose it as $A = LL^T$ where L is a lower triangular matrix. This is called the Cholesky factorization.

Theorem 3.5 says that LL^T is always positive definite. Cholesky factorization can be considered the other direction of Theorem 3.5.

Cholesky factorization is basically the LU factorization for PD matrices. We will still use the same matrix in Example 4.2 because it happens to be PD. We first do the regular LU decomposition, and get to

$$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 7 & 7 \\ 2 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 0 & 2 \end{bmatrix} = LU.$$

Currently these two matrices are not transpose of each other. To fix it, we can factor out 3 (row 2) and 2 (row 3) from the U matrix:

$$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 7 & 7 \\ 2 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & \sqrt{3} & 0 \\ 2 & \sqrt{3} & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & \sqrt{3} & \sqrt{3} \\ 0 & 0 & \sqrt{2} \end{bmatrix}$$

Cholesky factorization takes about $\frac{1}{3}n^3$ flops, half of the LU factorization due to symmetry.

4.4 Least Square

So far we have been dealing with the case that A is invertible, therefore $Ax = b$ has a unique solution. But real life systems often have no solutions due to many different reasons (too many unknowns, noise/perturbation, rounding off error...)

When $Ax = b$ does not have a solution, we seek the least square solution

$$x_{LS} = \arg\min_x \|Ax - b\|_2^2. \tag{4.4}$$

x_{LS} is called the *least square solution* of $Ax = b$. It can be proved (one way is to use geometrical approach using projections) that x_{LS} is the solution of the *normal equation*

$$A^T Ax = A^T b. \tag{4.5}$$

The normal equation is guaranteed to have one or infinitely many solutions for any matrix A.

When A has full column rank (columns are independent), then $A^T A$ is invertible hence one solution (left as an exercise), otherwise there are infinitely many solutions of (4.5). So x_{LS} can be mathematically expressed as $x_{LS} = (A^T A)^{-1} A^T b$, even though this is not how it's calculated in a computer. The expression of x_{LS} motivates us to define the *pseudoinverse* of A:

$$A^\dagger := (A^T A)^{-1} A^T.$$

Please note that the "simplification" $(A^T A)^{-1} A^T = A^{-1}(A^T)^{-1} A^T = A^{-1}$ is WRONG (why?), and only when A is invertible we have $A^\dagger = A^{-1}$.

There are many ways to solve the least square problem. We can come up with one quickly because the matrix $A^T A$ is positive definite so we can use Cholesky factorization.

Least Square via Cholesky factorization ($A \in \mathbb{R}^{m\times n}$ is full column rank, $m > n$)

Step 1:	Form $B = A^T A, f = A^T b$	$O(mn^2)$
Step 2:	Find Cholesky factorization $B = LL^T$.	$O(n^3)$
Step 2:	Solve $Ly = f$. (y will equal to $L^T x$.)	$O(n^2)$
Step 3:	Solve $L^T x = y$.	$O(n^2)$

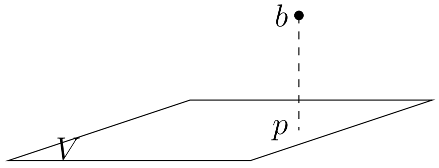


Figure 4.1: Projection onto V

Least Square problem is closely related to projections (default is orthogonal projections). Given a basis $\{a_1, a_2, \cdots, a_n\}$ of a subspace V, we let $A = [a_1, a_2, \cdots, a_n]$. If b is not on V (see Figure 4.1), then $Ax = b$ does not have a solution. Any vector in V can be represented by Ax for some x. By (4.4), Ax_{LS} is the point on V that is closest to b, which means

$$p = Ax_{LS}. \tag{4.6}$$

According to the calculation of x_{LS}, we have

$$p = AA^\dagger b = A(A^T A)^{-1} A^T b := Pb. \tag{4.7}$$

And

$$P = A(A^TA)^{-1}A^T \tag{4.8}$$

is called the Projection matrix onto V.

By (4.6) and (4.7), a second way to compute x_{LS} is to solve

$$Ax = Pb. \tag{4.9}$$

This will be useful later when we solve least square via QR decomposition or SVD.

When we have an orthonormal basis $Q = \{q_1, q_2, \cdots, q_n\}$ for V, then by Theorem 3.10 (c)

$$P = Q(Q^TQ)^{-1}Q^T = QQ^T \tag{4.10}$$

Remark 4.6. The projection matrix P is computed using a basis of V. There are infinitely many choices of bases of V, but no matter what basis you choose, you will end up having the same P.

Example 4.7. We have $b = (2,1,1)$ is not in $V = \text{span}\{(1,0,0),(0,1,0)\}$, because clearly that $Ax = b$ has no solution where $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$. Instead we solve the normal equation $A^TAx = A^Tb$, which becomes $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, so $x_{LS} = (2,1)$, and the projection of b onto V is $p = Ax_{LS} = (2,1,0)$. This makes sense because V is the xy-plane. The projection of any point onto the xy-plane is to set the z component to 0.

Example 4.8. Let $V = \text{span}\{u\}$. To find out the projection matrix onto this line V, we let $A = u$, so

$$P = u(u^Tu)^{-1}u^T = \frac{uu^T}{u^Tu} \tag{4.11}$$

Chapter 4 Exercises

1. How many **exact** flops are needed for the following computations?
   (a) $\langle x, y\rangle$, where x, y are vectors in $\mathbb{R}^9$.
   (b) ABC, where A, B, C are 5×5 matrices.
   (c) Ux, where U is $n \times n$ upper triangular matrix and x is a vector in $\mathbb{R}^n$.

2. Given $n \times n$ matrices A, B, C, how many flops are needed for the following computations? Answer in terms of the order $n, n^2, n^3, \cdots$.
   (a) ABC
   (b) $A^{-1}B$
   (c) $A + B$
   (d) LU, where L is $n \times n$ lower triangular, U is $n \times n$ upper triangular

3. Consider the linear system $\left[\begin{array}{ccccc} 2 & 2\times 10^{20} & \vdots & 2\times 10^{20} \\ 1 & 1 & \vdots & 2 \end{array}\right]$. Find its 'solution' using Gaussian elimination, in a machine where numbers are in standard IEEE double-precision format.

4. Solve $\begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} x = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$ $(Ax = b)$ by

   (a) First find the LU factorization of A.

   (b) Second solve $Ly = b$.

   (c) Third solve $Ux = y$.

5. Given $\begin{bmatrix} 3 & -6 & -3 \\ 2 & 0 & 6 \\ -4 & 7 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 \\ 2 & 4 & 0 \\ -4 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 & -2 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$, solve the system

$$\begin{array}{llll} 3x_1 & -6x_2 & -3x_3 & = -3 \\ 2x_1 & & +6x_3 & = -22 \\ -4x_1 & +7x_2 & +4x_3 & = 3 \end{array} .$$

6. Solve the system in Problem 5 using scaled partial pivoting strategy.

7. Let V be the coeffient matrix in (4.3). Find the permutation matrix P so that PV is upper triangular.

8. Compute the Cholesky factorization of $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$.

9. * If A has full column rank (columns are independent), then A^TA is invertible. (Hint: prove A^TA is positive definite. A Positive definite matrix is always invertible because its determinant is positive.)

10. Let $V = \text{span}\{a_1 = (1, -1, 0), a_2 = (0, 1, -1)\}$.

   (a) Find the projection matrix onto V using (4.8). Call it P_1.

   (b) Find an orthonormal basis $\{q_1, q_2\}$ for V. Let $Q = [q_1, q_2]$. Check that QQ^T is the same as P_1.

   (c) Find $V^\perp$. (This is the same as finding all vectors that are orthogonal to a_1, a_2.

   (d) Find the projection matrix onto $V^\perp$. Call it P_2.

   (e) Verify that $P_1 + P_2$ is the identity matrix.

Chapter 5

Singular Value Decomposition

A positive semidefinite (PSD) matrix is a symmetric matrix whose eigenvalues are nonnegative. If A is positive semidefinite, then A can be diagonalized as

$$A = QDQ^{-1} = QDQ^T, \tag{5.1}$$

where Q is an orthonormal matrix, and D is a diagonal matrix whose diagonals are the eigenvalues of A (hence all nonnegative).

Being positive semidefinite is a strong condition, and hence we can get a very nice decomposition (5.1). But the decomposition (5.1) is so nice that mathematicians want to do the same thing to every other matrix – impossible. When something is impossible, mathematicians do not give up. They find the next best possible thing, which is what we call the singular value decomposition (SVD).

Definition 5.1. Given any matrix A, whose size is $m \times n$, it can be proven that (we will not prove it here) A can be factored as

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

where U is an $m \times m$ orthonormal matrix, V is an $n \times n$ orthonormal matrix, and Σ is an $m \times n$ diagonal matrix. The entries on the diagonal of Σ are called the *singular values* of A. The singular values are always nonnegative. The singular value are often listed in descending order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}} \geq 0$. The columns of U and the columns of V are called the *left-singular vectors* and *right-singular vectors* of A respectively.

Example 5.2. $\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{2} & 0 \\ 0 & \boxed{0} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}^T.$

The singular values are 2,0.

Example 5.3. $\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \boxed{\sqrt{2}} & 0 \\ 0 & \boxed{\sqrt{2}} \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}^T.$

The singular values are $\sqrt{2}, \sqrt{2}$, which are quite different from its eigenvalues (they are complex numbers).

In general, given a square matrix, the singular values are very different from eigenvalues unless it is a symmetric matrix (see Example 5.6):

The singular values of a symmetric matrix are the absolute values of its eigenvalues. *In particular, the singular values of a PSD matrix are the same as its eigenvalues, because* (5.1) *would be its SVD.*

Example 5.4. $\begin{bmatrix} 0 & 0 & -2 & -2 \\ 1.5 & 1.5 & 2.5 & 2.5 \\ -3 & -3 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -1/3 & 2/3 & -2/3 \\ 2/3 & -1/3 & -2/3 \\ -2/3 & -2/3 & -1/3 \end{bmatrix} \begin{bmatrix} \boxed{6} & 0 & 0 & 0 \\ 0 & \boxed{3} & 0 & 0 \\ 0 & 0 & \boxed{0} & 0 \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 & -0.5 \end{bmatrix}^T.$

The singular values are 6,3,0.

It can be easily proven that **the rank of A will always equal to the number of positive singular values**. The columns of U and V also play an important role. Suppose A has rank r, then its nonzero singular values can be expressed descendingly as $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. The SVD can be expressed as

$$A = [u_1, u_2, \cdots, u_m] \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sigma_r & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} [v_1, v_2 \cdots, v_n]^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_r u_r v_r^T. \tag{5.3}$$

Make sure you understand that u_i is a column vector in $\mathbb{R}^m$ and v_i is a column vector in $\mathbb{R}^n$, so every $u_i v_i^T$ is an $m \times n$ matrix. Moreover, each $u_i v_i^T$ has rank 1 (prove it). So we are expressing A has the sum of r rank-1 matrices. For instance, the SVD in Example 5.4 can therefore be rewritten as

$$\begin{bmatrix} 0 & 0 & -2 & -2 \\ 1.5 & 1.5 & 2.5 & 2.5 \\ -3 & -3 & -1 & -1 \end{bmatrix} = 6 \begin{bmatrix} -1/3 \\ 2/3 \\ -2/3 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \end{bmatrix} + 3 \begin{bmatrix} 2/3 \\ -1/3 \\ -2/3 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 & -1/2 & -1/2 \end{bmatrix}.$$

Given that the rank of A is r, then we only need the first r columns of U and V in the reconstruction of A. Let $\hat{U} = [u_1, \cdots, u_r], \hat{V} = [v_1, \cdots, v_r]$, and $\hat{\Sigma}$ be the $r \times r$ upper left submatrix of Σ, then we have

$$A = \hat{U}\hat{\Sigma}\hat{V}^T \tag{5.4}$$

by (5.3). We can also use block matrix multiplication, which is left as an exercise. The decomposition (5.4) is called the *reduced SVD* of A, where both $\hat{U}, \hat{V}$ have orthonormal columns, and $\hat{\Sigma}$ is a square diagonal matrix.

The SVD of a matrix A can tell lots of useful information about A.

1. $\text{rank}(A) =$ [number of nonzero singular values of A] $= r$.

2. $C(A) = \text{span}\{u_1, u_2, \cdots, u_r\}$. Left-singular vectors is a basis of column space of A.

3. $R(A) = \text{span}\{v_1, v_2, \cdots, v_r\}$. Right-singular vectors is a basis of row space of A

4. $N(A) = \text{span}\{v_{r+1}, v_{r+2}, \cdots, v_n\}$.

Use Example 5.4 again, the matrix is rank 2, and its column space has a basis $\{u_1, u_2\} = \{\frac{1}{3}[-1, 2, -2]^T, \frac{1}{3}[2, -1, -2]^T\}$. Its row space has a basis $\{v_1, v_2\} = \{\frac{1}{2}[1, 1, 1, 1]^T, \frac{1}{2}[1, 1, -1, -1]^T\}$. Its null space has a basis $\{v_3, v_3\} = \{\frac{1}{2}[1, -1, -1, 1]^T, \frac{1}{2}[1, -1, 1, -1]^T\}$.

It is not an easy task to compute the SVD of A, but it is easy to compute its singular values. **For any matrix A, the singular values of A are the square roots of eigenvalues of A^TA (or AA^T).**

Example 5.5. Compute the singular values of $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -2 & 2 \end{bmatrix}$.

$A^T = \begin{bmatrix} 1 & 1 & -2 \\ 1 & 1 & 2 \end{bmatrix}$, $A^TA = \begin{bmatrix} 6 & -2 \\ -2 & 6 \end{bmatrix}$

$\begin{vmatrix} 6-r & -2 \\ -2 & 6-r \end{vmatrix} = (r-6)^2 - 4 = (r-6-2)(r-6+2) = (r-8)(r-4)$

Eigenvalues of A^TA are 8, 4.

Sinigular values of A are $\sqrt{8}, \sqrt{4}$.

If A is a short fat matrix, then you want to compute the eigenvalues of AA^T. Basically, you want to pick the smaller size out of A^TA and AA^T.

When A is symmetric, we can easily compute its SVD.

Example 5.6. Compute the SVD of $A = \begin{bmatrix} -7 & 6 \\ 6 & 2 \end{bmatrix}$.

Step 1: Find the eigenvalues and orthonormal eigenvectors of A

$r_1 = -10, u_1 = \frac{1}{\sqrt{5}}[-2, 1]^T$.

$r_2 = 5, u_2 = \frac{1}{\sqrt{5}}[1, 2]^T$.

Step 2: Diagonalize A with an orthonormal U.

$$A = UDU^T = \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} -10 & 0 \\ 0 & 5 \end{bmatrix} \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix}^T \tag{5.5}$$

Step 3: SVD

Equation (5.5) is not SVD because we need diagonals to be all positive. In order to get 10, we can simply change u_1 to $-u_1$.

$$A = \frac{1}{\sqrt{5}} \begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 10 & 0 \\ 0 & 5 \end{bmatrix} \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix}^T.$$

5.1 Rank-s Approximation

Given the following SVD,

$$A = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{\sqrt{20}} \\ \frac{1}{\sqrt{5}} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{\sqrt{20}} \\ \frac{1}{\sqrt{5}} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{\sqrt{20}} \\ \frac{1}{\sqrt{5}} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{\sqrt{20}} \\ \frac{1}{\sqrt{5}} & 0 & 0 & 0 & \frac{-4}{\sqrt{20}} \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}^T$$

We know that rank(A)=3, but it seems like A is almost rank 2 because the third singular value 0.01 can be neglected. Indeed, $A_2 = U \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$ is called the rank-2 approximation of A. After computation,

$$A = \begin{bmatrix} 3.52 & 2.81 & 0.005 \\ 3.52 & 2.81 & -0.005 \\ 2.81 & 3.52 & -0.005 \\ 2.81 & 3.52 & 0.005 \\ 3.16 & 3.16 & 0 \end{bmatrix}, A_2 = \begin{bmatrix} 3.52 & 2.81 & 0 \\ 3.52 & 2.81 & 0 \\ 2.81 & 3.52 & 0 \\ 2.81 & 3.52 & 0 \\ 3.16 & 3.16 & 0 \end{bmatrix}.$$

A_2 is very close to A as expected. The difference can sometimes be considered noise in applications. In combination with the earlier discussion, we can say that the columns of A are approximately on span$\{u_1, u_2\}$ = span$\{[1,1,1,1,1]^T, [1,1,-1,-1,0]^T\}$. The rows of A are approximately on span$\{v_1, v_2\}$ = span$\{[1,1,0]^T, [1,-1,0]^T\}$.

Notice the second singular value is also pretty small compared to 10, so we can approximate A by its rank-1 approximation $A_1 = U \begin{bmatrix} 10 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T = \begin{bmatrix} 3.16 & 3.16 & 0 \\ 3.16 & 3.16 & 0 \\ 3.16 & 3.16 & 0 \\ 3.16 & 3.16 & 0 \\ 3.16 & 3.16 & 0 \end{bmatrix}$. The error is bigger, but perhaps still acceptable in some circumstances. In this case, we can say that the columns of A are approximately on the line span$\{u_1\}$ = span$\{[1,1,1,1,1]^T\}$. The rows of A are approximately on the line span$\{v_1\}$ = span$\{[1,1,0]^T\}$.

In general, If $A = U \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sigma_r & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$, then given any $s < r$,

$$A_s = U \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_s & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} V^T = U\Sigma_s V^T = \sum_{i=1}^{s} \sigma_i u_i v_i^T$$

is called the closest rank-s approximation of A.

Example 5.7. Find the rank-1 approximation of the matrix in Example 5.6.

The SVD is already done, we just need to keep the biggest singular value 10. So rank-1 approximation is $\frac{1}{\sqrt{5}}\begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} 10 & 0 \\ 0 & 0 \end{bmatrix}\frac{1}{\sqrt{5}}\begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix}^T = \begin{bmatrix} -8 & 4 \\ 4 & -2 \end{bmatrix}$.

5.2 Data compression

This is a direct application of rank approximation.

Let D be an $m \times d$ data matrix, where each row is a data point. m and d might be both very big ($m > d$), which requires a lot of memory if we want to store D in a computer. A lot of times the data D is inherently in a lower dimension, and we can therefore store it with much less computer space, using SVD.

Let $D = \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_m^T \end{bmatrix}$

Step 1: Center all the data points at the origin.

Let $c = [c_1, c_2, \cdots, c_d]^T$, where c_i is the average of ith column of D, then c is the center of all m data points.

Let $E = \begin{bmatrix} p_1^T - c^T \\ p_2^T - c^T \\ \vdots \\ p_m^T - c^T \end{bmatrix}$. Now the rows (data points) of E are centered at 0.

Step 2: Take SVD of E: $E = U\Sigma V^T$. We are only interested in the V matrix because we want to compress rows of E. Let $V = [v_1, \cdots, v_d]$ and $\hat{V} = [v_1, \cdots, v_s]$.

Step 3: Observe the singular values, and we decide that the smallest $d - s$ ones are negligible.

This means that we will only keep the biggest s singular values, and the corresponding right-singular vectors $v_1, \cdots, v_s$.

Step 4: We store $\hat{V}$, $Y = E\hat{V}$, and c. Which has a total of $\boxed{ms + ds + d}$. entries.

The original D has $\boxed{md}$ entries, so this compression is significant if s is way smaller than d.

This is a loss compression. In order to get the data points back, we compute

$$E' = [Y, 0]V^T = Y\hat{V}^T, \text{ and } D' = E' + \begin{bmatrix} c^T \\ c^T \\ \vdots \\ c^T \end{bmatrix}.$$

D' will be an approximation of of the original D.

Justification: $E' = E[v_1, \cdots, v_s, 0, \cdots, 0]V^T = U\Sigma V^T[v_1, \cdots, v_s, 0, \cdots, 0]V^T = U\Sigma \begin{bmatrix} I_{s\times s} & 0 \\ 0 & 0 \end{bmatrix} V^T = U\Sigma_s V^T$

E' is the rank-s approximation of E.

Example 5.8. Suppose we have 5 points in $\mathbb{R}^3$. We write each point as a row of D

$$D = \begin{bmatrix} 3.4 & 2 & 6 \\ 3.4 & 5 & 0 \\ 0.4 & 2 & 3 \\ 0.4 & 2 & 6 \\ -2.6 & -1 & 0 \end{bmatrix}.$$

We first find the center of these 5 points: $c = [1, 2, 3]^T$. We make our E matrix:

$$E = \begin{bmatrix} 2.4 & 0 & 3 \\ 2.4 & 3 & -3 \\ -0.6 & 0 & 0 \\ -0.6 & 0 & 3 \\ -3.6 & -3 & -3 \end{bmatrix}.$$

We operate SVD on the matrix E.

$$E = \begin{bmatrix} -0.51 & -0.24 & 0.69 & 0.07 & 0.44 \\ -0.15 & 0.85 & -0.10 & 0.28 & 0.41 \\ 0.06 & -0.04 & -0.26 & -0.77 & 0.57 \\ -0.22 & -0.45 & -0.61 & 0.49 & 0.38 \\ 0.82 & -0.12 & 0.27 & 0.28 & 0.41 \end{bmatrix} \begin{bmatrix} 6.75 & 0 & 0 \\ 0 & 5.61 & 0 \\ 0 & 0 & 1.50 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.66 & 0.39 & 0.65 \\ -0.43 & 0.52 & -0.74 \\ -0.62 & -0.76 & -0.17 \end{bmatrix}^T$$

We notice that the third singular value is relatively insignificant compared to the other two. Let $V = [v_1, v_2, v_3]$. In a computer, instead of storing the whole D, we only store

$$Y = E[v_1, v_2] = \begin{bmatrix} -3.4371 & -1.3669 \\ -0.9983 & 4.7688 \\ 0.3935 & -0.2316 \\ -1.4697 & -2.5249 \\ 5.5117 & -0.6454 \end{bmatrix}, \hat{V} = [v_1, v_2], \text{ and } c.$$

Y looks nothing like D, but we can always get D back approximately as first compute

$$E' = Y\hat{V}^T = \begin{bmatrix} 1.7264 & 0.7694 & 3.1796 \\ 2.4955 & 2.8909 & -3.0255 \\ -0.3474 & -0.2885 & -0.0673 \\ -0.0108 & -0.6730 & 2.8429 \\ -3.8637 & -2.6988 & -2.9297 \end{bmatrix},$$

then add the center c back to each point (each row) of E', as

$$D' = \begin{bmatrix} 2.7264 & 2.7694 & 6.1796 \\ 3.4955 & 4.8909 & -0.0255 \\ 0.6526 & 1.7115 & 2.9327 \\ 0.9892 & 1.3270 & 5.8429 \\ -2.8637 & -0.6988 & 0.0703 \end{bmatrix}.$$

As we can see, the approximate D' is fairly close to the original D.
Notice that E' is the rank-2 approximate of E as we justified earlier.

$$E_2 = \begin{bmatrix} -0.51 & -0.24 \\ -0.15 & 0.85 \\ 0.06 & -0.04 \\ -0.22 & -0.45 \\ 0.82 & -0.12 \end{bmatrix} \begin{bmatrix} 6.75 & 0 \\ 0 & 5.61 \end{bmatrix} \begin{bmatrix} -0.66 & 0.39 \\ -0.43 & 0.52 \\ -0.62 & -0.76 \end{bmatrix}^T = \begin{bmatrix} 1.7264 & 0.7694 & 3.1796 \\ 2.4955 & 2.8909 & -3.0255 \\ -0.3474 & -0.2885 & -0.0673 \\ -0.0108 & -0.6730 & 2.8429 \\ -3.8637 & -2.6988 & -2.9297 \end{bmatrix}.$$

Chapter 5 Exercises

1. Verify the computation in (5.3) using block matrix multiplication. (U matrix is partitioned into $1 \times m$ blocks (columns), and V is partitioned into $n \times 1$ blocks (rows).)

2. Let U be $m \times m$, V be $n \times n$, and $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sigma_r & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}_{m \times n}$. Let $\hat{U}$ be the $m \times r$ matrix consisting of the first r columns of U, and $\hat{V}$ be the $n \times r$ matrix consisting of first r columns of V. Show that $U\Sigma V^T = \hat{U} \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_r \end{bmatrix} \hat{V}^T$ using block matrix multiplication.

3. Compute the singular values of $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -2 & 2 \end{bmatrix}$.

4. The singular value decomposition of B is

$$B = \frac{1}{\sqrt{2}} \begin{bmatrix} \sin\frac{\pi}{4} & \sin\frac{2\pi}{4} & \sin\frac{3\pi}{4} \\ \sin\frac{2\pi}{4} & \sin\frac{4\pi}{4} & \sin\frac{6\pi}{4} \\ \sin\frac{3\pi}{4} & \sin\frac{6\pi}{4} & \sin\frac{9\pi}{4} \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0.02 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 \end{bmatrix}^T$$

(a) What is the V matrix in svd of B?

(b) What is the rank of B

(c) Find a basis of $C(B)$.

(d) Find a basis of $N(B)$?

(e) *On which line do the columns of B approximately lie? On which line do the rows of B approximately lie? (Easy problem!)

5. Let $A = \frac{1}{2} \begin{bmatrix} 0 & 3 & -6 \\ 0 & 3 & -6 \\ -4 & 5 & -2 \\ -4 & 5 & -2 \end{bmatrix}$. Use Example 5.4 to find the reduced SVD of A.

Chapter 6

Stability and conditioning

In the abstract, we can view a problem as a function $f : X \longrightarrow Y$ from a vector space X to another vector space Y. x is the input and $y = f(x)$ can be viewed as the solution of this problem. **Stability or conditioning of a problem** means how much the solution (output) y changes given a perturbation on x. A *well-conditioned* problem is one with the property that all small perturbations of x lead to only small changes in $f(x)$. An *ill-conditioned* problem is one with the property that some small perturbation of x leads to a large change in $f(x)$.

For example, f could be a univariate scalar-valued function, more of which can be found in Section 6.1.

We particularly care more about solving problems in linear algebra. We want to explore the stability of solving linear systems $Ax = b$, in which case $f = A^{-1}$ and $X = Y = \mathbb{R}^n$. This is further illustrated in Section 6.2. Two other problems that are very important in linear algebra are finding least square solution and finding eigenvalues, and we care about their stability very much in practice. The problem of finding eigenvalues of a nonsymmetric matrix is often ill-conditioned. One can see this by comparing the two matrices

$$\begin{bmatrix} 1 & 1000 \\ 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} 1 & 1000 \\ 0.001 & 1 \end{bmatrix},$$

whose eigenvalues are $\{1, 1\}$ and $\{0, 2\}$, respectively.

We also very much care about the **stability of an algorithm** for a particular problem. This stability is induced by the rounding off error in floating point number representations. Section 6.4 talks about three different algorithms for solving the least square problem, one of which is to use the QR decomposition, which is covered in Section 6.3.

A lot of the material here comes from [2], if readers are interested in more details.

6.1 One variable to one variable function

Let $f(x)$ be a scalar-valued function. If x is close to $\hat{x}$, how close is $y = f(x)$ to $\hat{y} = f(\hat{x})$? For example, consider the square root function $\sqrt{x}$. Let $x = 0.0001$ and $\hat{x} = 0.00011$ be relatively close to x, then $y = 0.01$ and $\hat{y} \approx 0.0105$. The difference in y is about 5×10^{-4}, although seemingly small, it is 50 times the difference in x (10^{-5}).

In the example above, we are considering in an absolute sense: If

$$|\hat{y} - y| = C(x)|\hat{x} - x|,$$

then $C(x)$ is called the *absolute condition number* of the function f at x.. We also can ask the question in a relative sense: If

$$\left|\frac{\hat{y}-y}{y}\right| = \kappa(x)\left|\frac{\hat{x}-x}{x}\right|,$$

then $\kappa(x)$ is called the relative condition number of f at x.

We want the condition numbers to be small.

The condition numbers can be estimated using derivatives.

$$C(x) = \left|\frac{f(\hat{x})-f(x)}{\hat{x}-x}\right| \approx |f'(x)|$$

$$\kappa(x) = \left|\frac{f(\hat{x})-f(x)}{\hat{x}-x}\cdot\frac{x}{f(x)}\right| \approx \left|\frac{xf'(x)}{f(x)}\right|$$

Example 6.1. In the previous square root example, $C(x) = |(\sqrt{x})'| = \frac{1}{2}x^{-1/2}$, which can be large when x is small as shown above. But the relative condition number $\kappa(x) = |x\cdot 0.5x^{-1/2}/x^{1/2}| = 0.5$ is good. So we can say that this function is well conditioned in a relative sense.

6.2 Stability of solving linear system

Here we are concerned with the behavior of the solution of $Ax = b$ when b is perturbed.

Given the following linear system

$$\begin{bmatrix} 101 & 99 \\ 99 & 101 \end{bmatrix} x = \begin{bmatrix} 200 \\ 200 \end{bmatrix}, \tag{6.1}$$

we can solve it (even in head) and the answer is $x = [1, 1]^T$.

Suppose now we perturb the right hand side of (6.1) a little bit and solve

$$\begin{bmatrix} 101 & 99 \\ 99 & 101 \end{bmatrix} \hat{x} = \begin{bmatrix} 202 \\ 198 \end{bmatrix} \tag{6.2}$$

instead. One would hope that $\hat{x}$ is not far from x, however, a simple calculation concludes that $\hat{x} = [2, 0]^T$. This unfortunately looks drastically different from the original x. We can say that the linear system $\begin{bmatrix} 101 & 99 \\ 99 & 101 \end{bmatrix}$ is not very stable, and we will come back to this example after proper definitions.

Suppose A is invertible, and we are trying to solve x from $Ax = b$. However, we get a noised $\hat{b}$ and let $\hat{x}$ be the solution to the linear system $A\hat{x} = \hat{b}$. We have $A(x-\hat{x}) = b - \hat{b}$. By matrix norm property vi (section 3.5),

$$\|x-\hat{x}\| \le \|A^{-1}\|\cdot\|b-\hat{b}\|$$

We are more interested in the relative error, so

$$\frac{\|x-\hat{x}\|}{\|x\|} \le \frac{\|A^{-1}\|\cdot\|b-\hat{b}\|}{\|x\|} = \frac{\|A^{-1}\|\cdot\|Ax\|}{\|x\|}\frac{\|b-\hat{b}\|}{\|b\|} \le \|A^{-1}\|\cdot\|A\|\cdot\frac{\|b-\hat{b}\|}{\|b\|} := \kappa(A)\frac{\|b-\hat{b}\|}{\|b\|}.$$

We often use the operator norm (2-norm) in the computing of condition number, although other norms have been used as well.

$$\kappa(A) = \|A^{-1}\|_2 \cdot \|A\|_2$$

is defined as the condition number of a matrix A.

Theorem 6.2. *If A is invertible, then $\|A^{-1}\|_2$ is the reciprocal of the smallest singular value of A.*

Proof 1. We can use the singular value decomposition. $A = U\Sigma V^T$, and let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ be the diagonal entries of Σ. We can denote Σ by $\Sigma = \text{diag}(\sigma_1, \cdots, \sigma_n)$. Then

$$A^{-1} = (U\Sigma V^T)^{-1} = (V^T)^{-1}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^T.$$

This means $V\Sigma^{-1}U^T$ is the SVD of A^{-1} because V, U are orthonormal and $\Sigma^{-1} = \text{diag}(\sigma_1^{-1}, \cdots, \sigma_n^{-1})$. Consequently the singular values of A^{-1} are $\sigma_1^{-1}, \cdots, \sigma_n^{-1}$.

$\|A^{-1}\|_2 = \max\{\sigma_1^{-1}, \cdots, \sigma_n^{-1}\} = \sigma_n^{-1}$.

Proof 2. Proof 1 uses SVD, which is too powerful of a tool. We present a more elementary proof here, where we use the fact that **eigenvalues of B^{-1} are inverse of eigenvalues of B.** (This is easy to prove as if $Bx = \lambda x$, then $x = \lambda B^{-1}x \Longrightarrow B^{-1}x = \lambda^{-1}x$.)

$\|A^{-1}\|_2^2 =$ (biggest singular value of $A^{-1})^2 =$ biggest eigenvalue of $A^{-T}A^{-1}$
$=$ biggest eigenvalue of $(A^TA)^{-1} = 1/($smallest eigenvalue of $A^TA)$
$= 1/($smallest singular value of $A)^2$ □

This directly implies the following theorem, which is how we compute condition numbers.

Theorem 6.3. *Given an invertible $n \times n$ matrix A with singular values $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_n > 0$*

$$\kappa(A) = \frac{\sigma_1}{\sigma_n}.$$

This means that condition number of a matrix is always greater than or equal to 1. A matrix is best conditioned when the condition number is 1, and ill-conditioned when the condition number is huge.

Example 6.4. Let A be the coefficient matrix in (6.1), then $\det[A - \lambda I] = (101 - \lambda)^2 - 99^2 = (101 - \lambda - 99)(101 - \lambda + 99)$, so the eigenvalues are 2 and 200. Due to symmetry, singular values are —2— and —200—, so $\kappa(A) = 200/2 = 100$.

Example 6.5. Any orthonormal matrix has condition number 1.

6.3 QR decomposition and its algorithms

The QR decomposition is the most important and useful decomposition in numerical linear algebra. We first start with Gram-Schmidt which is related.

Review on Gram-Schmidt

Given $\{a_1, a_2, \cdots, a_n\}$ independent, how to find $\{q_1, q_2, \cdots, q_n\}$ such that

1. $\{q_1, q_2, \cdots, q_n\}$ is an orthonormal set, and

2. $\text{span}\{q_1, q_2, \cdots, q_n\} = \text{span}\{a_1, a_2, \cdots, a_n\}$?
One common method is to use the Gram-Schmidt algorithm.

Gram-Schmidt

$$q_1 = \frac{a_1}{r_{11}} \qquad r_{11} = \|a_1\|_2$$
$$q_2 = \frac{a_2 - r_{12}q_1}{r_{22}}$$
$$q_3 = \frac{a_3 - r_{13}q_1 - r_{23}q_2}{r_{33}} \qquad r_{ij} = q_i^T a_j$$
$$\vdots$$
$$q_n = \frac{a_n - \sum_{i=1}^{n-1} r_{in}q_i}{r_{nn}} \qquad r_{jj} = \|a_j - \sum_{i=1}^{j-1} r_{ij}q_i\|_2$$

This algorithm can be rearranged as

$$\begin{aligned}
a_1 &= r_{11}q_1\\
a_2 &= r_{12}q_1 + r_{22}q_2\\
a_3 &= r_{13}q_1 + r_{23}q_2 + r_{33}q_3\\
&\cdots\\
a_n &= r_{1n}q_1 + r_{2n}q_2 + r_{3n}q_3 + \cdots + r_{nn}q_n,
\end{aligned}$$

which is equivalent to

$$[a_1, a_2, \cdots, a_n] = [q_1, q_2, \cdots, q_n] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & r_{n-1,n} \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix} \tag{6.3}$$

Example 6.6. Use the Gram-Schmidt algorithm to construct a set of orthonormal vectors spanning the same space as $\text{span}\{a_1 = (1, -1, 1, -1), a_2 = (1, 0, 1, 0), a_3 = (4, 0, 2, -2)\}$.

$$q_1 = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}, r_{11} = 2$$

$$r_{12} = q_1^T a_2 = 1, q_2 = \frac{1}{r_{22}}\left(\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \|r_{22}\| = 1.$$

$r_{13} = q_1^T a_3 = 4, r_{23} = q_2^T a_3 = 2,$

$$q_3 = \frac{1}{r_{33}}\left(\begin{bmatrix} 4 \\ 0 \\ 2 \\ 2 \end{bmatrix} - \frac{4}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} - \frac{2}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}\right) = \frac{1}{r_{33}}\left(\begin{bmatrix} 4 \\ 0 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ -2 \\ 2 \\ -2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

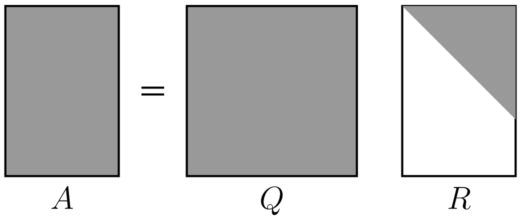
$r_{33} = 2$.
This means

$$\begin{bmatrix} 1 & 1 & 4 \\ -1 & 0 & 0 \\ 1 & 1 & 2 \\ -1 & 0 & -2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}\begin{bmatrix} 2 & 1 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{bmatrix}. \tag{6.4}$$

Given any matrix $A \in \mathbb{R}^{m\times n}$, we can decompose A as

$$A = QR,$$

where Q is an $m \times m$ orthonormal matrix and R is an $m \times n$ upper triangular matrix. Below is a visualization when $m \geq n$.

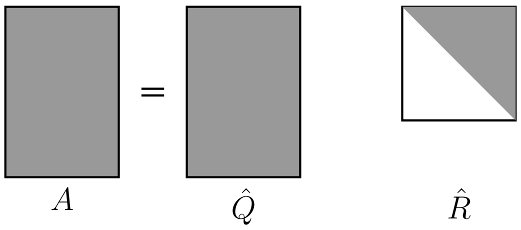


This is also called the *full QR factorization.*

Let $\hat{Q}$ be the first n columns of Q, and $\hat{R}$ be the first n rows of R, then

$$QR = [\hat{Q}, *]\begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} = \hat{Q}\hat{R} + 0 = \hat{Q}\hat{R}.$$

$A = \hat{Q}\hat{R}$ is the *reduced QR decomposition* of A, where $\hat{Q}$ has orthonormal columns, and $\hat{R}$ is a square upper triangular matrix. See below.



Gram-Schmidt algorithm is one way to find the QR decomposition. For example, the equation (6.4) is the reduced QR decomposition. The full QR is left as an exercise. However, Gram-Schmidt is not the most stable algorithm for this problem. We introduce a better algorithm below. It is called the Househouder triangulation. We first introduce/review three concepts: orthogonal complement, hyperplane, and Householder reflector.

Given a subspace V, the *orthogonal complement* of V is defined as the collection of vectors that is orthogonal to everything in V. It is denoted as $V^\perp$

$$V^\perp = \{x : \langle x, y\rangle = 0 \quad \forall\, y \in V\}. \tag{6.5}$$

For example, $(\text{span}\{(0,0,1)\})^\perp = \text{span}\{(1,0,0),(1,2,0)\}$. The orthogonal complement of the z-axis is the xy-plane. It is easy to check that $(V^\perp)^\perp = V$.

To find out the orthogonal complement of a given subspace, definition (6.5) askes to find vectors that is orthogonal to every vector in V. Fortunately, **we only need to find vectors that is orthogonal to the basis vectors of** V (why?). For example, if we know a subspace V_2 in $\mathbb{R}^3$ has a basis $\{(1,1,1),(1,1,-1)\}$, then we only need to find vectors orthogonal to (1,1,1) and (1,1,-1), which boils down to find the null space of the matrix whose rows are these two vectors. $V_2^\perp = \text{null space of } \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$.

Below is a very important theorem on orthgonal complement. First, for any subspace U, P_U denotes the projection matrix onto U (see (4.8)).

Theorem 6.7. *Given a subspace $V \subset \mathbb{R}^d$, then*

$$P_V + P_{V^\perp} = I_{d\times d}.$$

This theorem can provide an alternative and possibly easier way to compute projection matrices.

Example 6.8. Let $V = \text{span}\left\{ v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \end{bmatrix} \right\}$, find P_V.

Method 1: Let $A = [v_1, v_2, v_3]$. $P_V = A(A^TA)^{-1}A^T$. This method could be very computationally heavy and is not recommended for this problem.

Method 2: We first find $V^\perp$. This is to solve $A^Tx = 0$, and we get $V^\perp = \text{span}\{v_4 = (0,0,0,1)\}$. By (4.11),

$$P_{V^\perp} = \frac{v_4 v_4^T}{v_4^T v_4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and by Theorem 6.7

$$P_V = I_{4\times 4} - P_{V^\perp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

In linear algebra, a *plane* usually refers to a 2-dimensional subspace in $\mathbb{R}^3$. This fits the English word "plane" except that a subspace has to go through origin. a *hyperplane* generalizes this concept to higher dimensional Euclidean space.

Definition 6.9. A subspace $S \subset \mathbb{R}^d$ is called a hyperplane if $\dim(S) = d - 1$.

For example, V in Example 6.8 is a hyperplane because the vectors are in $\mathbb{R}^4$ and $\dim(V) = 3$.

By the definition of a hyperplane, the dimension of $S^\perp$ is always 1 (due to fundamental theorem of linear algebra: dim(row space) + dim(null space) = d), so the characterization of a hyperplane is

$$S \text{ is a hyperplane} \iff S = (\text{span}\{u\})^\perp \text{ for some vector } a.$$

This characterization can be again seen in Example 6.8 where $u = v_4$.

Definition 6.10. Let H_u be a matrix such that $H_u x$ is the reflection of x with respect to the hyperplane $S = (\text{span}\{u\})^\perp$. The operator/matrix H_u is called a *Householder reflector.*

Figure 6.1 shows the reflection. Note that the vector u only indicates a direction. $H_u = H_{cu}$ for any scalar $c \neq 0$.

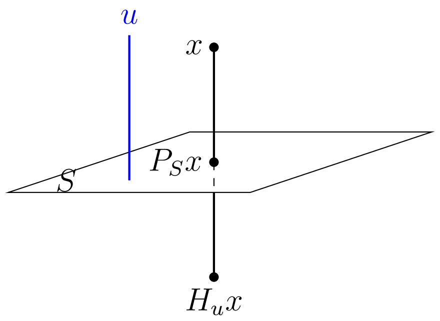


Figure 6.1: Reflection with respect to $S = (\text{span}\{u\})^\perp$

It is not hard to compute H_u. Since $P_S x$ is the middle point of x and $H_u x$, we have $\dfrac{x + H_u x}{2} = P_S x \implies H_u x = 2P_S x - x$. Since this is true for every vector x, then $H_u = 2P_S - I$ (see Exercise 3 of this chapter). $P_S = I - \dfrac{uu^T}{u^T u}$ due to Theorem 6.7 and (4.11), so

$$H_u = I - 2\frac{uu^T}{u^T u} \tag{6.6}$$

It is easy to check that H_u is symmetric and orthonormal (left as an exercise).

In this chapter $\|\cdot\|$ is the 2-norm $\|\cdot\|_2$.

Example 6.11. Given $x = (x_1, x_2)$, find u such that $H_u x = \|x\|(1, 0)$.

We can use simple geometry to figure out u as shown in Figure 6.2. First we draw a dashed line, across which x is reflected to $\|x\|e_1$. This dashed line is the hyperplane. By the definition of H_u, we need to find a vector perpendicular to the dashed line. Since the vectors x and $\|x\|e_1$ have the same length, simply connecting these two vectors (blue line) will give us a vector perpendicular to the dashed line. We pick $u = x - \|x\|e_1$. (We can also pick any scalar multiple of $x - \|x\|e_1$.)

To double check such u will lead to $H_u x = \|x\|e_1$, we compute $u^T u = \langle x - \|x\|e_1, x - \|x\|e_1\rangle = \langle x, x\rangle - 2\langle x, \|x\|e_1\rangle + \langle \|x\|e_1, \|x\|e_1\rangle = \|x\|^2 - 2\|x\|x_1 + \|x\|^2 = 2\|x\|^2 - 2\|x\|x_1$.

$u^T x = \langle x - \|x\|e_1, x\rangle = \|x\|^2 - \|x\|x_1$.

So $H_u x = (I - 2\dfrac{uu^T}{u^T u})x = x - 2\dfrac{uu^T x}{u^T u} = x - \dfrac{2u^T x}{u^T u}u = x - u = \|x\|e_1$ as desired.

Moreover, we can find $u' = x + \|x\|e_1$ such that $H_{u'} x = -\|x\|e_1$.

Example 6.11 is our basis for the second method of QR decomposition. In general, for vectors in $\mathbb{R}^d$, we have

$$\text{If } u = x \pm \|x\|e_1, \text{ then } H_u x = \mp\|x\|e_1 \tag{6.7}$$

The equation (6.7) should be read as if $u = x + \|x\|e_1$, then $H_u x = -\|x\|e_1$; if $u = x - \|x\|e_1$, then $H_u x = \|x\|e_1$.

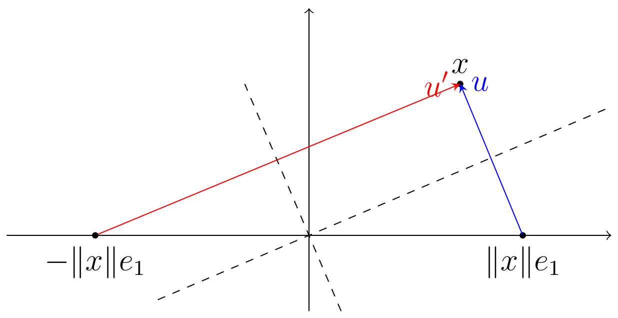


Figure 6.2: Reflect to x-axis

Now we are ready to introduce QR decomposition by Householder triangularization. Recall that in LU decomposition, we left multiply A by a sequence of L's as $L_K \cdots L_2 L_1 A = U$, where each L represents a row operation. L's are lower triangular but not orthonormal. In QR decomposition, we want to left multiply A by a sequence of orthonormal matrices Q_i to make it upper triangular. The matrix Q_i is chosen to introduce zeros below the diagonal in the ith column while preserving all the zeros previously introduced. For example, in the 5×3 case, three Q_i's are applied as follows:

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{Q_2} \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & 0 & \times \\ & 0 & \times \\ & 0 & \times \end{bmatrix} \xrightarrow{Q_3} \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \times \\ & & 0 \\ & & 0 \end{bmatrix}$$

$$A_0 := A \qquad A_1 := Q_1 A \qquad A_2 := Q_2 Q_1 A \qquad A_3 := Q_3 Q_2 Q_1 A$$

From A to $Q_1 A$, inspired by Example 6.11, we simply let $Q_1 = H_u$ such that H_u transfers the first column a to $\|a\|e_1$.

Moreover, we require that applying Q_2 does not destroy the zeros in the first column, and preserve the first row of $Q_1 A$. Similarly, applying Q_3 does not destroy the zeros in the previous two columns, and preserve the first two rows of $Q_2 Q_1 A$. This means u has two choices: $a + \|a\|e_1$ or $a - \|a\|e_1$.

To achieve this, we choose Q_k to be an orthonormal matrix of the form (see Exercise 7 of this chapter)

$$Q_k = \begin{bmatrix} I_{(k-1)\times(k-1)} & 0 \\ 0 & H_{u_k} \end{bmatrix},$$

where

$$u_k = a_k \pm \|a_k\|e_1, \text{ and } a_k \text{ is the } k\text{th column of } A_{k-1} \text{ starting at } k\text{th row.} \tag{6.8}$$

The Householder reflector is used in the construction of Q_k because it can transform a_k to $\pm(\|a_k\|, 0, \cdots, 0)$, thus introduce zeros below the first entry.

In the end, the R matrix is A_K (if there are K columns), and

$$Q_K \cdots Q_2 Q_1 A = R \Longrightarrow A = (Q_1^T Q_2^T \cdots Q_K^T) R.$$

Example 6.12. Compute the QR decomposition of $A_0 = \begin{bmatrix} 1 & 1 & 4 \\ -1 & 0 & 0 \\ 1 & 1 & 2 \\ -1 & 0 & -2 \end{bmatrix}$ using Householder triangularization. This A_0 is the same matrix as in Example 6.6.

Step 1: $a_1 = (1, -1, 1, -1)$, $u_1 = a_1 - \|a_1\|e_1 = (1, -1, 1, -1) - (2, 0, 0, 0) = (-1, -1, 1, -1)$. (We could also choose $u_1 = a_1 + \|a_1\|e_1$.)

$$Q_1 = H_{u_1} = I_{4\times 4} - 2\frac{u_1 u_1^T}{u_1^T u_1} = \frac{1}{2}\begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}, \qquad A_1 = Q_1 A = \begin{bmatrix} 2 & 1 & 4 \\ 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & -2 \end{bmatrix}.$$

Step 2: $a_2 = (0, 1, 0), u_2 = a_2 - \|a_2\|e_1 = (0, 1, 0) - (1, 0, 0) = (-1, 1, 0)$.

$$H_{u_2} = I_{3\times 3} - 2\frac{u_2 u_2^T}{u_2^T u_2} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & H_{u_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_2 = Q_2 A_1 = \begin{bmatrix} 2 & 1 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & -2 \end{bmatrix}.$$

Step 3: $a_3 = (0, -2), u_3 = a_3 - \|a_3\|e_1 = (-2, -2)$.

$$H_{u_3} = I_{2\times 2} - 2\frac{u_3 u_3^T}{u_3^T u_3} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}, Q_3 = \begin{bmatrix} I_{2\times 2} & 0 \\ 0 & H_{u_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

$$A_3 = Q_3 A_2 = \begin{bmatrix} 2 & 1 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}.$$

In the end, $Q = Q_1^T Q_2^T Q_3^T, R = A_3$.

Remark 6.13. We allow two choices for u_k in (6.8). In theory, both are valid choices. In practice, we choose $u_k = a_k + \text{sign}(\text{first coordinate of } a_k)\|a_k\|e_1$ for stability (due to rounding off). This can be briefly explained using figure 6.2. In that picture, we should be choosing $x + \text{sign}(x_1)\|x\|e_1 = x + \|x\|e_1$: the red u'. Why is u' better than u for stability? Imagine x has a very tiny first component (like (0.01, 1)), then the blue vector u will be very small, whereas u' is large. Small number tends to create problems in general.

With this said, we should have chosen $u_1 = a_1 + \|a_1\|e_1$ in Step 1 of Example 6.12 (for stability reason) because the first coordinate of a_1 is positive.

6.4 Three algorithms for the Least square problem

Suppose we have $A \in \mathbb{R}^{m\times n}$ that is full column rank. There are three common algorithms for solving the least square solution of $Ax = b$. The first one uses Cholesky factorization to solve the normal equation $A^T A x = A^T b$. It is introduced in Section 4.4.

The second algorithm uses QR factorization. The third algorithm uses SVD deomposition.

Both uses the equation (4.9) $Ax = Pb$, where P is the projection matrix onto the $V = C(A) =$ column space of A.

If we already have the reduced QR factorization $A = \hat{Q}\hat{R}$, then $P = \hat{Q}\hat{Q}^T$. There are two ways to think about this. First one is to use (4.8), which is straightforward (left as an exercise). Second way is to realize that $C(\hat{Q}) = C(A) = V$, and then use (4.10) since columns of $\hat{Q}$ are orthonormal. Think about (6.4) in Example 6.6, where the columns of A and $\{q_1, q_2, q_3\}$ span the same space.

So now $Ax = Pb$ can be changed to $\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^T b$. Left multiplying $\hat{Q}^T$, we get $\hat{R}x = \hat{Q}^T b$ (use Theorem 3.10 (c)), which can be solved easily by backward substitution since $\hat{R}$ is upper triangular.

Least Square via QR factorization ($A \in \mathbb{R}^{m\times n}$ **is full column rank,** $m > n$)

Step 1:	Compute the reduced QR decomposition $A = \hat{Q}\hat{R}$	$O(mn^2)$
Step 2:	Compute $\hat{Q}^T b$.	$O(mn)$
Step 3:	Solve $\hat{R}x = \hat{Q}^T b$.	$O(n^2)$

The SVD method is similar. If we already have the reduced SVD $A = \hat{U}\hat{\Sigma}V^T$ (here rank$(A) = n$, so $\hat{V} = V$), then $P = \hat{U}\hat{U}^T$. The equation $Ax = Pb$ is converted to $\hat{U}\hat{\Sigma}V^T x = \hat{U}\hat{U}^T b$. Left multiplying both sides by $\hat{U}^T$, we get $\hat{\Sigma}V^T x = \hat{U}^T b$. We first solve $\hat{\Sigma}y = \hat{U}^T b$, which only takes n operations because $\hat{\Sigma}$ is an $n \times n$ diagonal matirx. We then solve $V^T x = y$, which has a straightforward solution $x = Vy$ because V is an orthonormal matrix.

Least Square via SVD ($A \in \mathbb{R}^{m\times n}$ **is full column rank,** $m > n$)

Step 1:	Compute the reduced SVD $A = \hat{U}\hat{\Sigma}V^T$	$O(mn^2)$
Step 2:	Compute $\hat{U}^T b$.	$O(mn)$
Step 3:	Solve $\hat{\Sigma}y = \hat{U}^T b$.	$O(n)$
Step 4:	$x = Vy$	$O(n^2)$

Comparison of algorithms:

Each of the methods we described is advantageous in certain situations. When speed is the only consideration, Cholesky factorization may be the best. However, solving the normal equations is not always stable in the presence of the rounding errors, and thus numerical analysts have recommended the QR factorization route for "daily use". If A is close to rank-deficient, however, the QR method has less-than-ideal stability properties, and in such cases there are good reasons to turn to the SVD approach.

Chapter 6 Exercises

1. $A = \begin{bmatrix} 101 & 99 \\ 99 & 101 \end{bmatrix}$ has condition number 100, but it doesn't mean that solution is always sensitive to perturbation, for example, let $b = (2, -2)$ and $\hat{b} = (2.01, -2)$. Let $x, \hat{x}$ be the

solutions as $Ax = b$ and $A\hat{x} = \hat{b}$. Compute the number C such that $\dfrac{\|x - \hat{x}\|_2}{\|x\|_2} = C\dfrac{\|b - \hat{b}\|_2}{\|b\|_2}$, using a calculator or computer.

2. Find the full QR decomposition of $\begin{bmatrix} 1 & 1 & 4 \\ -1 & 0 & 0 \\ 1 & 1 & 2 \\ -1 & 0 & -2 \end{bmatrix}$ using the reduced one (6.4).

3. Let A be an $n \times n$ matrix. Prove that if $Ax = 0$ for every vector $x \in \mathbb{R}^n$, then A must be the zero matrix.

4. Show that H_u in (6.6) is symmetric and orthonormal.

5. Show that $H_{-2u} = H_u$.

6. Given $x = (1, 2, -2)$,

   (a) find an orthonormal matrix Q such that $Qx = 3e_1$.

   (b) find an orthonormal matrix Q' such that $Q'x = -3e_2$. (Hint: read first paragraph of Example 6.11.)

   (c) can you find an orthonormal matrix Q'' such that $Q''x = 2e_1$? Why?

7. Let U be an $n \times n$ matrix. It is in the partitioned form as $U = \begin{bmatrix} I_{k\times k} & 0 \\ 0 & B \end{bmatrix}$, where B is an $(n-k) \times (n-k)$ orthonormal matrix. Show that

   (a) The first k rows of UA is the same as the first k rows of A using block matrix multiplication.

   (b) U is orthonormal using block matrix multiplication.

8. Redo Example 6.12 where Step 1 is kept, but in Step 2 and Step 3, we let $u_i = a_i + \|a_i\|e_1$.

9. If $A = QR$ is its full QR decomposition, what is the Cholesky factorization of A^TA?

10. Let A be full column rank and P be the projection matrix onto the column space of A.

    (a) If we have the reduced QR factorization as $A = \hat{Q}\hat{R}$, then prove that $P = \hat{Q}\hat{Q}^T$.

    (b) If we have the reduced SVD as $A = \hat{U}\hat{\Sigma}V^T$, then prove that $P = \hat{U}\hat{U}^T$.

11. Given the reduced QR factorization (6.4), find the least square solution of

$$\begin{bmatrix} 1 & 1 & 4 \\ -1 & 0 & 0 \\ 1 & 1 & 2 \\ -1 & 0 & -2 \end{bmatrix} x = \begin{bmatrix} 2 \\ -1 \\ 0 \\ 1 \end{bmatrix}$$

    using "least square via QR" algorithm.

12. Given the SVD in Chapter 5 Exercise 4,

(a) First find the reduced SVD of B^T.

(b) Find the least square solution of $B^T x = (2, -1, 0, 1)$ using the "least square via SVD" algorithm. Have the trigs calculated and do not use a calculator.

Chapter 7

Polynomial Interpolation

Given $n+1$ points in $\mathbb{R}^2$: $\{(x_i, y_i)\}_{i=0}^n$, any function that goes through these $n+1$ points is called an *interpolant* of these points. In this chapter, we wish to find a polynomial that can go through these points **exactly**. This is different from least square problems as we are fitting these data points exactly, rather than approximately.

This is a classical topic from approximation theory, and we will only present some important theories here. We often use p_n for a polynomial of degree n.

Computers can do additions and multiplications really well, therefore making polynomials the most friendly function to a computer. For this reason, we would like to interpolate a given set of data points with polynomials.

In order to examine how good of an interpolation job a polynomial does, we need to formally define the distance between two functions, which is some kind of norm of the difference of two functions mathematically. Suppose we are looking at

$$C[a,b] := \text{ the vector space of all continuous functions on the interval } [a,b].$$

For $f \in C[a,b]$, define the L_∞ norm of f as

$$\|f\|_\infty := \max_{x\in[a,b]} |f(x)|.$$

Define the L_p norm ($p \geq 1$) of f as

$$\|f\|_p := \left(\int_a^b |f(x)|^p dx\right)^{1/p}.$$

We use the L_2 norm the most often, which is $\|f\|_2 := \left(\int_a^b |f(x)|^2 dx\right)^{1/2}$.

Example 7.1. Let $f(x) = x$ and $g(x) = e^x$ on $[0,1]$.

$\|f\|_\infty = 1, \|g\|_\infty = e, \|f\|_2^2 = \int_0^1 x^2 dx = 1/3.$

To compute $\|f-g\|_2$ (the L_2 distance between f and g), we first compute the integral $\int_0^1 (x-e^x)^2 dx = \int_0^1 x^2 - 2xe^x + e^{2x} dx = \frac{1}{3} + \frac{e^{2x}}{2}\Big|_0^1 - 2\int_0^1 xe^x dx \overset{\text{integration by parts}}{=} \frac{1}{3} + \frac{e^2-1}{2} - 2(xe^x - e^x)\Big|_0^1 = \frac{1}{3} + \frac{e^2-1}{2} - 2(0+1) = \frac{3e^2-13}{6}.$

Then $\|f-g\|_2 = \sqrt{\frac{3e^2-13}{6}} \approx 1.236.$

7.1 General polynomial interpolation

We can find a line going through two points, and we can find a quadratic going through 3 given points. In general, it is believable that we are able to find a polynomial of degree n to fit $n+1$ points because such polynomial has $n+1$ coefficients ($n+1$ degrees of freedom) and we have exactly $n+1$ pieces of information.

Theorem 7.2. *Given $\{(x_i, y_i)\}_{i=0}^n$ where the $n+1$ values of x_i are all different, then there is a unique polynomial p of degree at most n that goes through these $n+1$ points.*

The proof is easy. Subsection 7.1.1 can be viewed as a proof.

We will introduce 3 well-known methods for finding this p.

7.1.1 Vandermonde matrix

Let $p(x) = \sum_{i=0}^{n} c_i x^i$. In order to figure out the coefficients c_i, we plug in the known points (x_j, y_j), which generates $n+1$ equations $\sum_{i=0}^{n} c_i x_j^i = y_j, j = 0, 1, \cdots, n$. These equations are all linear, so we get an $(n+1)\times(n+1)$ system

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}. \tag{7.1}$$

This is a straightforward method. The coefficient matrix in the system (7.1) is in the form of a Vandermonde matrix.

In general, a $k \times k$ *Vandermonde matrix* is a matrix generated by a vector $v = (v_1, v_2, \cdots, v_k)$ in the following fashion:

$$\text{For } 1 \le i \le k, \text{ the } i\text{th column is } [v_1^{i-1}, v_2^{i-1}, \cdots, v_k^{i-1}]^T.$$

It can be proven that as long as $x_i \neq x_i$ for any $i \neq j$, then the Vandermonde matrix is invertible, thus (7.1) has a unique solution. In the solution, c_n (or any other coefficient) could be 0, therefore the polynomial we get is of degree at most n.

Example 7.3. Find the quadratic that goes through the three points $(-1, 0), (0, 1), (2, 1)$.

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 2 & 4 & 1 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -1 & 1 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 3 & 3 & 1 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -1 & 1 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 6 & -2 \end{array}\right]$$

$c_2 = -1/3; c_1 - (-1/3) = 1 \Rightarrow c_1 = 2/3; c_0 - 2/3 - 1/3 = 0 \Rightarrow c_0 = 1.$

$y = 1 + \frac{2}{3}x - \frac{1}{3}x^2.$

7.1.2 Lagrange interpolation

Lagrange solved this problem in a different approach, where no linear system is involved. First we generate $n+1$ polynomials $\varphi_i(x)$ such that

$$\varphi_i(x_j) = \delta_{ij} := \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} .$$

δ_{ij} is a useful notation and is heavily used in mathematics and engineering.

The construction of φ_i is immediate. For example, if $x_0 = 2, x_1 = 2.5, x_2 = 4$, then φ_0 has roots 2.5 and 4, so $\varphi_0(x) = c(x-2.5)(x-4)$. c is determined by $\varphi_0(x_0) = 1$, so we can write $\varphi_0(x) = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)}$. Similarly, $\varphi_1(x) = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)}$ and $\varphi_2(x) = \frac{(x-2)(x-2.5)}{(4-2)(4-2.5)}$.

In general, for $\{x_i, y_i\}_{i=0}^n$

$$\varphi_i(x) = \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}. \tag{7.2}$$

Note that φ_i only depends on x_i.

The second step of the Lagrange method is even easier, the answer that we are looking for is then

$$p(x) = \sum_{i=0}^{n} y_i \varphi_i(x) \tag{7.3}$$

Example 7.4. Redo Example 7.3 using Lagrange's method.

$x_0 = -1, x_1 = 0, x_2 = 2$.

Form $\varphi_0(x) = \frac{(x-0)(x-2)}{(-1-0)(-1-2)}, \varphi_1(x) = \frac{(x+1)(x-2)}{(0+1)(0-2)}, \varphi_2(x) = \frac{(x+1)(x-0)}{(2+1)(2-0)}$. After simplification, we have $\varphi_0(x) = \frac{1}{3}x(x-2), \varphi_1(x) = \frac{-1}{2}(x+1)(x-2), \varphi_2(x) = \frac{1}{6}x(x+1)$.

So $p(x) = 0\cdot\varphi_0(x) + 1\cdot\varphi_1(x) + 1\cdot\varphi_2(x) = \frac{1}{6}(x+1)[-3(x-2)+x] = \frac{1}{3}(x+1)(-x+3)$.

7.1.3 Newton's interpolation

The Newton's interpolation is similar to the first method. We only formulate $p(x)$ differently as

$$p(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n\prod_{i=0}^{n-1}(x-x_i).$$

Plugging in $y_0 = p(x_0)$ implies $y_0 = a_0$. Plugging in $y_1 = p(x_1)$ implies $y_1 = a_0 + a_1(x_1 - x_0)$... In the end, we get a lower triangular system

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & x_1-x_0 & 0 & 0 & \cdots & 0 \\ 1 & x_2-x_0 & (x_2-x_0)(x_2-x_1) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 1 & x_n-x_0 & (x_n-x_0)(x_n-x_1) & \cdots & & \prod_{i=0}^{n-1}(x_n-x_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{7.4}$$

It is very efficient to solve (7.4) because it is a lower triangular system.

Example 7.5. Redo Example 7.3 using Newton's interpolation.

We get a different system $\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 1 & 0+1 & 0 & 1 \\ 1 & 2+1 & (2+1)(2-0) & 1 \end{array}\right]$, solving it we get $a_0 = 0, a_1 = 1, a_2 = -1/3$.

So $p(x) = (x+1) - \dfrac{1}{3}(x+1)(x-0)$.

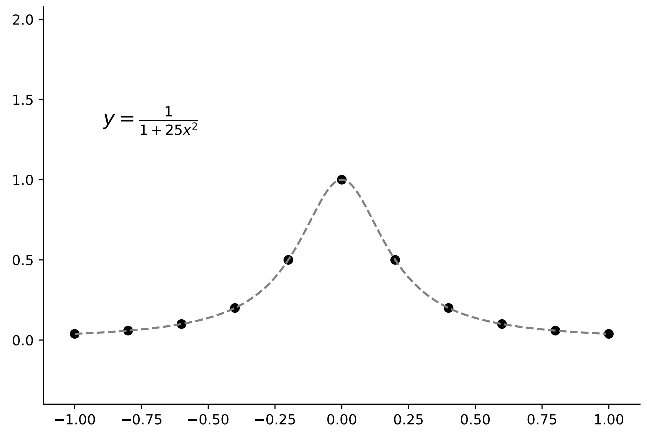
One benefit of using Newton's formulation is that the computation is minimal for adding a data point. Suppose now we are trying to find the degree 3 polynomial going through $(-1,0),(0,1),(2,1)$ and $(1.5,3)$. The system will become

$$\left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0 \\ 1 & 0+1 & 0 & 0 & 1 \\ 1 & 2+1 & (2+1)(2-0) & 0 & 1 \\ 1 & 1.5+1 & (1.5+1)(1.5-0) & (1.5+1)(1.5-0)(1.5-2) & 3 \end{array}\right].$$

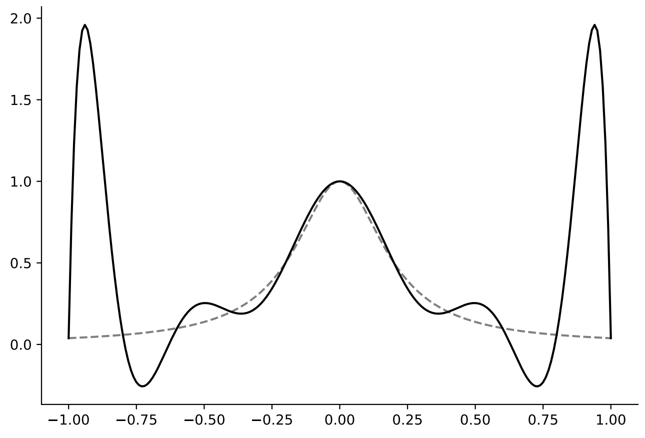
We can reuse the answers a_0, a_1, a_2 solved earlier because they are the same equations. We only need to solve for a_3 as the additional work.

7.1.4 Runge function effect

Consider the function $R(x) = \dfrac{1}{1+25x^2}$ on the interval $I = [-1,1]$. This is called the Runge function (scaled). Suppose we are interpolating 11 equally spaced points that we get from sampling $R(x)$ (Figure 7.1 (a)). Figure 7.1 (b) is the graph of the 10 degree polynomial that goes through these 11 points. Call this polynomial $P(x)$. Notice that there are big oscillations near the ends of the interval. In fact, as degree increases, the oscillations near the end will get even larger.



(a) Runge function



(b) Interpolate at 11 equally spaced points

Figure 7.1: Runge function effect

The L_2 difference between R and P is

$$\|R-P\|_2 = \sqrt{\int_0^1 (R(x)-P(x))^2\,dx} = 0.821.$$

We introduce two approaches to reduce this runge function effect. The first approach is to not pick equally spaced points. This is in Section 7.2. The second approach, discussed in Section 7.3, is to use multiple lower degree polynomials.

7.2 Chebyshev nodes

The following theorem illustrates the point-wise error of polynomial interpolation.

Theorem 7.6. *Assume that $f \in C^{n+1}[a,b]$ ($n+1$th derivative is continuous) and that $x_0, \cdots, x_n$ are in $[a,b]$. Let $p(x)$ be the polynomial of degree n that interpolates f at $\{x_i\}_{i=0}^n$. Then for any point $x \in [a,b]$,*

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{7.5}$$

for some point $\xi \in [a,b]$.

The proof only needs Rolle's theorem, see Theorem 8.4.1 of [1] for details.

This gives us an estimate on the L_∞ distance between f and p on the interval [-1,1]:

$$\|f - p\|_\infty = \max_{x\in[-1,1]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \prod_{i=0}^{n} |x - x_i| \le \max_{x\in[-1,1]} \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \max_{x\in[-1,1]} \prod_{i=0}^{n} |x - x_i|.$$

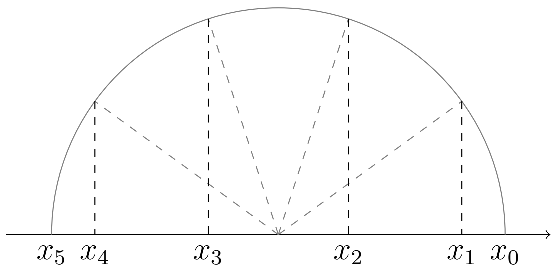
Suppose f is given (like the runge function), in order to minimize the error bound, we need to solve

$$\min_{\{x_0,\cdots,x_n\}\subset[-1,1]} \max_{x\in[-1,1]} \prod_{i=0}^{n} |x - x_i|. \tag{7.6}$$

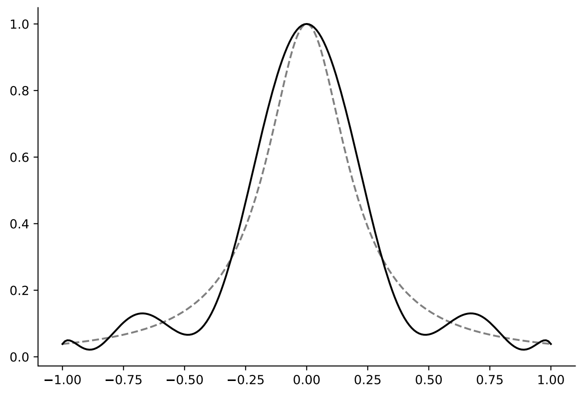
The solution (not trivial) for this minimization problem is

$$\text{Chebyshev nodes:} \quad x_i = \cos\frac{\pi i}{n}, i = 0, 1, \cdots, n \tag{7.7}$$

which are the x coordinates of n equally spaced angles on the unit circle, see Figure 7.2 (a). These are the roots of the Chebyshev polynomial of the first kind. They are not equally spaced, and are more clustered around the ends.



(a) Chebyshev nodes for $n = 5$



(b) Interpolation at Chebyshev nodes for $n = 10$

Figure 7.2: Interpolation with Chebyshev nodes

Figure 7.2 (b) is the Chebyshev interpolant, which approximates the runge function much better. We call this polynomial $C(x)$. The L_2 error $\|R - C\|_2 = 0.088$, is only about 1/10 of the previous one. Therefore if there is a choice on where to interpolate, picking the Chebyshev nodes is optimal.

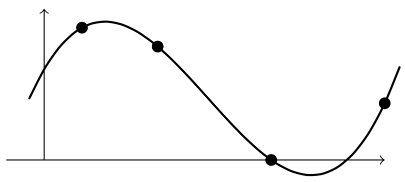
The definition $x_i = \cos\dfrac{\pi i}{n}$ only works for the interval [-1,1]. What happens if we have [-2,2], or in general $[a, b]$? This is left as an exercise.

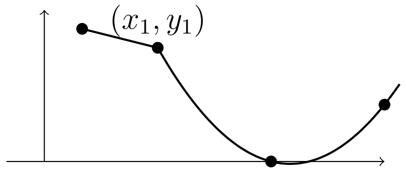
7.3 Spline

Spline means piecewise polynomials, meaning it is a piecewisely defined function and on each piece it is a different polynomial. Moreover, we generally require it is continuous. This simply means that different pieces connect at the same point. There is no jump or drop. The following function is an example of spline.

$$s(x) = \begin{cases} -x^2 + x, & 0 \le x \le 1 \\ x^3, & 1 \le x \le 2 \\ 2x + 4, & 2 \le x \le 3 \end{cases}$$

The purpose of spline is to interpolate the same number of data points, with lower degrees, at the cost of multiple polynomials. Lower degree polynomials are not only faster to be evaluated computers, they also tend to have less oscillations, which potentially reduces the Runge function effect. Figure 7.3 (a) is to interpolate 4 points with a degree 3 polynomial (cubic), and Figure 7.3 (b) is to interpolate the first two points with a line (degree 1) and the last three with a quadratic (degree 2).


(a) One polynomial interpolation



(b) Spline interpolation: a not desirable case

Figure 7.3

However, Figure 7.3 (b) is a situation that we want to avoid. It is not smooth at the second point x_1. To be precise, this piecewise function is not differentiable at x_1 because the left derivative, which is the slope of the line, does not coincide with the right derivative.

Because spline is multiple polynomials piecing together, we need to make sure they are "well-connected" at x_i's. In real applications, we often require it to be twice differentiable. Below is an example of a (one time) differentiable spline.

Example 7.7. Given three points $(0, 1), (1, -1), (2, 0)$, find an interpolant

$$s(x) = \begin{cases} P_1(x), & 0 \le x \le 1 \\ P_2(x), & 1 \le x \le 2 \end{cases}$$

such that $s(x)$ is differentiable. P_1 is a polynomial of degree 1, and P_2 is a polynomial of degree 2.

This boils down to
(1) $P_1(0) = 1, P_1(1) = -1; P_2(1) = -1, P_2(2) = 0$
(2) $P_1'(1) = P_2'(1)$.

Condition (1) already completely determines P_1: $P_1(x) = -2x+1$. Therefore by (2) we know $P_2'(1) = P_1'(1) = -2$. Since P_2 is of degree 2, we can let $P_2'(x) = 2a(x-1)-2$, which means that $P_2(x) = a(x-1)^2 - 2x + b$. We use condition (1) to determine a, b: $-1 = -2+b, 0 = a-4+b$. So we get $a = 3, b = 1$. In the end, $P_1(x) = -2x+1, P_2(x) = 3(x-1)^2 - 2x + 1$. It is graphed in Figure 7.4.

In order to simply fulfill the continuity requirement (1), there could be many other choices of P_2 (like the gray dashed lines in Figure 7.4), but only the black one connects with the line smoothest.

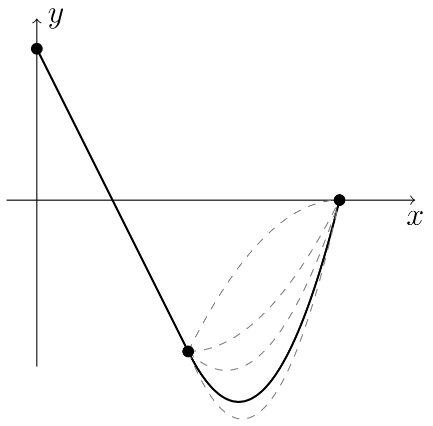


Figure 7.4: Spline interpolation: a somewhat desirable case

If a function is differentiable in higher order, then it is considered to be "smoother". Let us consider another piecewise defined function $f(x) = q_2(x) := -x^2+x$ on [0,1] and $f(x) = q_3(x) := (x-1)(x^2-3x+1)$ on [1,2], see Figure 7.5. The quadratic q_2 and the cubic q_3 share the 0th, 1st, and 2nd derivative at $x = 1$ as $q_2(1) = q_3(1) = 0, q_2'(1) = q_3'(1) = -1, q_2''(1) = q_3''(1) = -2$. In fact, there are infinitely many other cubics will do this without specifying the value at $x = 2$ (left as an exercise). As a result, $f(x)$ is twice differentiable, and it transits even smoother at $x = 1$ than $s(x)$ in Example 7.7.

To show the difference between smoothness of once differentiable and twice differentiable functions, we draw in Figure 7.5 some other cubic pieces that will have the same first derivative as q_2 at $x = 1$, but fails to have the second derivative equal (in gray dashed lines). Notice that the second derivative reveals the concavity of a function. In Figure 7.5, q_3 is keeping the concavity near 1, whereas the gray dashed lines are changing the concavity.

Next section will specifically talk about how to use multiple degree 3 polynomials (cubic) to interpolate given points smoothly (twice differentiable).

7.3.1 Cubic Spline

Spline has the most applications in 2D and 3D computer graphics where one wishes to design a smooth curve/surface based on a collection of nodes. There are numerous examples including design of fonts, car manufacture, and animation.

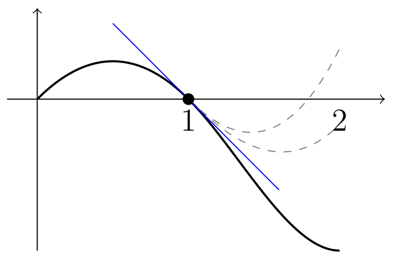
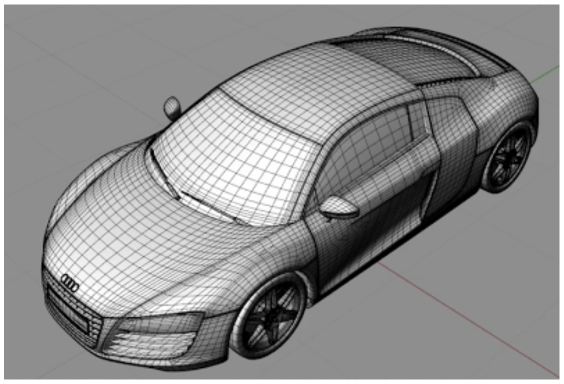


Figure 7.5: Twice differentiable spline: a desirable case



Given $\{(x_i, y_i)\}_{i=0}^n$, a Cubic spline is to find n cubics $q_i(x)$ such that the interpolant

$$s_c(x) = \begin{cases} q_1(x), & x_0 \le x \le x_1 \\ q_2(x), & x_1 \le x \le x_2 \\ \vdots \\ q_n(x) & x_{n-1} \le x \le x_n \end{cases}$$

is twice differentiable.

According to the discussion earlier, this precisely means

(a) Each cubic goes through two points: $q_i(x_{i-1}) = y_{i-1}, \quad q_i(x_i) = y_i, \quad i = 1, 2, \cdots, n$

(b) Derivative match: $q_i'(x_i) = q_{i+1}'(x_i), \quad i = 1, 2, \cdots, n-1$

(c) Second derivative match: $q_i''(x_i) = q_{i+1}''(x_i) = z_i, \quad i = 1, 2, \cdots, n-1$. ($z_i$ to be determined.)

Each cubic has 4 coefficients, so there are a total of $4(n+1)$ unknowns. We get $2n + n - 1 + n - 1 = 4n - 2$ constraints from (a)(b)(c). This leaves 2 degrees of freedom, which can be used to enforce various conditions on the endpoints of the spline, resulting in different types of cubic splines.

It is not hard to derive a formula for s_c, but for simplicity, we let x_i's to be equally spaced, so every subinterval has length $h = \dfrac{b-a}{n} = \dfrac{x_n - x_0}{n}$. We start with constraint (c): For $i = 1, \cdots, n$, q_i'' is a line, and it goes through (x_{i-1}, z_{i-i}) and (x_i, z_i), by Lagrange's method, we get

$$i = 1, ..., n \qquad q_i''(x) = z_{i-1}\frac{x - x_i}{x_{i-1} - x_i} + z_i\frac{x - x_{i-1}}{x_i - x_{i-1}} = -\frac{1}{h}z_{i-1}(x - x_i) + \frac{1}{h}z_i(x - x_{i-1}). \tag{7.8}$$

Integrating (7.8) twice, we obtain

$$i = 1, ..., n \qquad q_i'(x) = -\frac{1}{2h}z_{i-1}(x - x_i)^2 + \frac{1}{2h}z_i(x - x_{i-1})^2 + C_i. \tag{7.9}$$

$$i = 1, ..., n \qquad q_i(x) = -\frac{1}{6h} z_{i-1}(x - x_i)^3 + \frac{1}{6h} z_i(x - x_{i-1})^3 + C_i(x - x_{i-1}) + D_i. \tag{7.10}$$

Constraint (a) generates $\begin{cases} -\frac{1}{6h} z_{i-1}(-h)^3 + D_i = y_{i-1} \\ \frac{1}{6h} z_i(h)^3 + C_i h + D_i = y_i \end{cases}$, so

$$D_i = y_{i-1} - \frac{h^2}{6} z_{i-1}, \qquad C_i = \frac{1}{h}\left(y_i - y_{i-1} + \frac{h^2}{6}(z_{i-1} - z_i)\right) \tag{7.11}$$

Plugging in C_i in (7.9),

$$q_i'(x) = -\frac{1}{2h} z_{i-1}(x - x_i)^2 + \frac{1}{2h} z_i(x - x_{i-1})^2 + \frac{1}{h}\left(y_i - y_{i-1} + \frac{h^2}{6}(z_{i-1} - z_i)\right).$$

Finally we use constraint (b) to derive $n - 1$ linear equations about z_i:

$$i = 1, ..., n - 1, \quad \frac{h}{6} z_{i-1} + \frac{2h}{3} z_i + \frac{h}{6} z_{i+1} = \frac{1}{h} y_{i-1} - \frac{2}{h} y_i + \frac{1}{h} y_{i+1}. \tag{7.12}$$

There are $n - 1$ equations, and $n + 1$ constraints, so we are free to choose z_0 and z_n. The choice $z_0 = z_n = 0$ gives what is called the natural cubic spline, in which case (7.12) becomes a symmetric tridiagonal system

$$\begin{bmatrix} \frac{h}{6} & \frac{2h}{3} & & & \\ \frac{2h}{3} & \frac{h}{6} & \frac{2h}{3} & & \\ & \frac{2h}{3} & \frac{h}{6} & \ddots & \\ & & \ddots & \ddots & \frac{2h}{3} \\ & & & \frac{2h}{3} & \frac{h}{6} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} \frac{1}{h} y_0 - \frac{2}{h} y_1 + \frac{1}{h} y_2 \\ \frac{1}{h} y_1 - \frac{2}{h} y_2 + \frac{1}{h} y_3 \\ \vdots \\ \vdots \\ \frac{1}{h} y_{n-2} - \frac{2}{h} y_{n-1} + \frac{1}{h} y_n \end{bmatrix}. \tag{7.13}$$

Example 7.8. Find the natural cubic spline that interpolates $(0, 0), (1, 0), (2, 2), (3, 2), (4, -1)$.

$h = 1$, (7.12) becomes $\begin{bmatrix} 1/6 & 2/3 & 1/6 & 0 & 0 \\ 0 & 1/6 & 2/3 & 1/6 & 0 \\ 0 & 0 & 1/6 & 2/3 & 1/6 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ -3 \end{bmatrix}$.

We choose $z_0 = 0, z_4 = 0$, and we get a 3×3 system $\begin{bmatrix} 2/3 & 1/6 & 0 \\ 1/6 & 2/3 & 1/6 \\ 0 & 1/6 & 2/3 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ -3 \end{bmatrix}$ whose solution is $z_1 = 15/4, z_2 = -3, z_3 = -15/4$. We can use (7.10) and (7.11) to compute q_i. For example, $D_1 = y_0 - z_0/6 = 0, C_1 = y_1 - y_0 + (z_0 - z_1)/6 = -5/8$, so

$q_1(x) = \frac{z_1}{6}(x - x_0)^3 + C_1(x - x_0) + D_1 = \frac{5}{8}x^3 - \frac{5}{8}x.$

Exercises

1. Let $f(x) = 3/2 - x/2, g(x) = \dfrac{1}{x}$. Both functions belong to $C[1,2]$. Find
   (a) $\|f - g\|_2$
   (b) $\|f - g\|_\infty$ (use derivative to find max)

2. Find the polynomial interpolant of (1,7/4), (2,3/2), (3.5,0), and (5,3/4) by
   (a) Vandermonde method (feel free to use python)
   (b) Lagrange method (handwritten, don't simplify)

3. Use python to compute the condition number of the following two Vandermonde matrices, and comment on the results briefly.
   (a) generated by $0, 0.2, 0.4, 0.6, 0.8$.
   (b) generated by $0, 0.2, 0.22, 0.6, 0.8$.

4. Given $P_1 = (-1, 0), P_2 = (0, -1), P_3 = (1, 1), P_4 = (2, 1)$. Find the polynomial interpolant going through
   (a) P_2, P_3, P_4
   (b) P_1, P_2, P_3, P_4

   (Think about which method you should use given the connection of part (a) and part (b))

5. (a) The Chebyshev nodes defined in (7.7) is only for interval [-1,1]. What do you do if the interval is [-2,2], or $[a, b]$ in general?

   (b) Write down the Chebyshev nodes for the interval [-1,3]. $n = 6$.

6. * Solve (7.6) for the simplest case $n = 1$. That is to solve $\min\limits_{\{x_0,x_1\}\subset[-1,1]} \max\limits_{x\in[-1,1]} |x-x_0||x-x_1|$.

7. Determine an interpolant of (0,1), (1,0), (2,0) P such that

$$P(x) = \begin{cases} P_2(x)(\text{degree } 2), & 0 \le x \le 1 \\ P_1(x)(\text{degree } 1), & 1 \le x \le 2 \end{cases}$$

   and P is differentiable on [0,2].

8. If $s(x) = -x^2 + x$ when $0 \le x \le 1$ and $s(x) = P_3(x)$ on $1 \le x \le 2$. Find all possible cubic P_3 that makes $s(x)$ twice differentiable on [0,1].

9. Finish Example 7.8 by computing the rest q_2, q_3, q_4. Plot this cubic spline.

Chapter 8

Optimization

An *optimization problem*, has the form

$$\begin{array}{ll} \min & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, i = 1, 2 \cdots, m \end{array} . \tag{8.1}$$

Here $x = (x_1, \cdots, x_n)$ is multivariable in general, the function f_0 is called the *objective* function and $f_i, i = 1, \cdots, m$ are the constraint functions. Both objective and constraint functions are scalar-valued functions.

Example 8.1. Minimize $f(x) = x^2$ on the interval $[-2, 3]$. This is a univariate function. $f_0(x) = x^2$. There are two restrictions $x - 3 \leq 0, -x - 2 \leq 0$, so $f_1(x) = x - 3$ and $f_2(x) = -x - 2$.

Example 8.2. Problem (7.6) is an optimization problem, where $x = (x_0, x_1, \cdots, x_n)$, $f_0(x) = \max\limits_{t\in[-1,1]} \prod\limits_{i=0}^{n} |t - x_i|$, and there are $2n + 1$ constraints $-1 \leq x_i \leq 1$, which can be converted to $f_i(x) \leq 0$ easily.

Example 8.3. The least square problem $\min\limits_{x} \|Ax - b\|_2$ is an optimizetion problem where there is no constraints.

We need to use the following Calculus III concepts in this chapter.

Given a multivariate function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. The *gradient* of f at x is a vector, denoted as $\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n})$. The gradient can be thought as the first derivative of f. The *Hessian* of f is a matrix

$$Hf = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

The Hessian is like "the second derivative" of f. The functions considered here will be nice enough that $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$. Therefore the Hessian is a symmetric matrix.

Example 8.4. $f(x) = x_1^2 + x_1x_2 + x_2^2$. Then $\dfrac{\partial f}{\partial x_1} = 2x_1 + x_2$, $\quad \dfrac{\partial f}{\partial x_2} = x_1 + 2x_2$.

$\nabla f = (2x_1 + x_2, x_1 + 2x_2)$. $Hf = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

Example 8.5. $g(x) = x_1^2 + 2x_1x_2^2$. Then $\dfrac{\partial g}{\partial x_1} = 2x_1 + 2x_2^2$, $\quad \dfrac{\partial g}{\partial x_2} = 4x_1x_2$.

$\nabla g = (2x_1 + 2x_2^2, 4x_1x_2)$. $Hg = \begin{bmatrix} 2 & 4x_2 \\ 4x_2 & 4x_1 \end{bmatrix}$.

8.1 Convex optimization

A *convex optimization* problem is an optimization problem where the objective function f_0 and the constraint functions f_i are all convex functions. (8.1) is a nonconvex optimization problem if any of the f_i is not a convex function. Numerical analysts have developed theories, tools, and algorithms to solve different convex optimization problems, and we will learn a couple important ones here. In general,

> *If you have formulated an optimization problem to a convex optimization problem, then you have solved the problem.*

So what is a convex function? You may have learned about concave up/down functions in Calculus I. For one variable, a concave up function, like $f(x) = x^2$ is a convex function. We often use the second derivative to test the convexity. If $f''(x) \geq 0$ on a domain I, then we can claim that f is convex on I. But the definition of convexity does not rely on the existence of derivatives.

For a multivariable function f, it is a *convex function* on a convex domain I (see (8.4) for what is a convex set) if

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y), \text{ for every } x, y \in I, \theta \in [0,1]. \tag{8.2}$$

The quantity $\theta x + (1-\theta)y$ is a weighted average of x and y, so the definition (8.2) means that f evaluated at the average is less than the average function value. We draw a graph of univariate convex function in Figure 8.1. The equation (8.2) translates to "The point B is higher than the point A.", meaning a line segment of $(a, f(a))$ and $(b, f(b)$ is above the graph of f between (a, b).

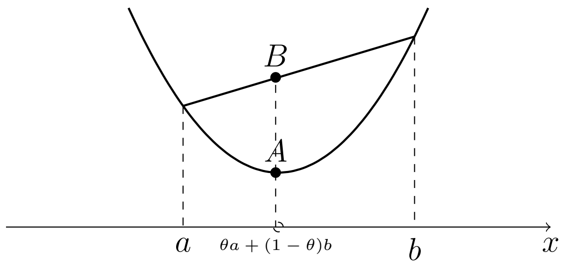


Figure 8.1: The definition of a convex function

If the inequality in (8.2) becomes "=", then such function is linear, as $f(x_1, \cdots, x_n) = a_1x_1 + \cdots + a_nx_n + b$ or $f(x) = a^Tx + b$. Clearly a linear function is a special case of convex function. However, in convex optimization, we would like to avoid the linear/flat case and seek strict convexity. f is *strictly convex* if it is convex and the equality in (8.2) only holds when $x = y$. Strict convexity assures one global minimizer, which is an ideal situation to be in.

We can also use the second derivative – the Hessian matrix to determine the convexity of f, given that the second partial derivatives exist.

Theorem 8.6. *Given $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and its Hessian matrix Hf on a domain I. If Hf is positive definite on I, then f is strictly convex on I. If Hf is positive semidefinite on I, then f is convex on I.*

In Example 8.4, Hf is positive definite everywhere, so f is strictly convex everywhere. In Example 8.5, we need $8x_1 - 16x_2^2 > 0$ to ensure positive definiteness, so we can claim that g is convex on $I = \{(x_1, x_2) : x_1 > 2x_2^2\}$.

Example 8.4 is an important case. We can see that f is a quadratic as $f = x^T \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} x$, and further more $\nabla f = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x$. So $f, \nabla f$ and Hf are all related to the same matrix. We summarize it below.

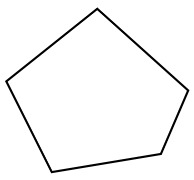
Theorem 8.7. *If $q(x) = \frac{1}{2}x^T Ax + a^T x + b$ (quadratic term + linear term + constant term), then $\nabla q = Ax + a$, $Hq = A$. Consequently, q is a strictly convex function if A is positive definite, and q is convex if A is positive semidefinite.*

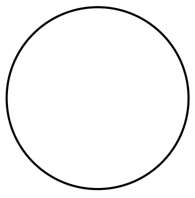
Another equivalent way to define a convex problem is

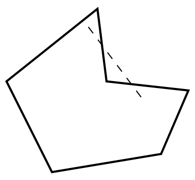
$$\begin{array}{ll} \min & f_0(x) \\ \text{subject to} & x \in C \end{array} \tag{8.3}$$

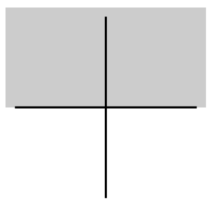
where f_0 is a convex function and C is a convex set. This is because $x \in C$ can always be rewritten as $\{x : f_i(x) \leq 0, i = 1, \cdots, m\}$ for convex functions f_i, and vice versa. See Exercise 3 for a relevant problem.

Geometrically, a convex set/region/doamin is a set where there is no place that is caved in. See the four examples in $\mathbb{R}^2$ below. The fourth example is an unbounded region.


(a) Convex set


(b) Convex set


(c) Nonconvex set


(d) upper half plane is convex

Formally speaking, a set C is convex if

$$\text{For any } x, y \in C, \text{ any } \theta \in [0, 1], \quad \theta x + (1 - \theta)y \text{ is still in } C. \tag{8.4}$$

For fixed x, y, $\{\theta x + (1 - \theta)y : \theta \in [0, 1]\}$ is exactly the line segment between x and y, so (8.4) can be interpreted as "Any line segment of points in C should remain in C." Clearly picture (c) above violates this statement. The definition (8.4) is also to serve for the definition of a convex function (8.2) as to ensure that $\theta x + (1 - \theta)y$ is still in the domain of f.

8.2 Unconstrained convex optimization

We will introduce two methods for solving unconstrained convex optimization problems

$$\min_x f(x) \tag{8.5}$$

where f is convex everywhere. Both algorithms are iterative and aim to find the optimal solution x^*. Convexity guarantees that there are only one local minimum, which is the global minimum. To find this minimum, we just need to set all the partial derivatives to be 0, i.e., $\nabla f = 0$.

8.2.1 The Gradient Descent Method

Geometrically, a convex function has a bowl shape. If we start anywhere, we would like to slide down to the bottom. The gradient of a function has such functionality. If we move towards the opposite direction of the gradient, then we are moving to the bottom (minimum). This is the idea of the gradient descent method. This method is the number 1 used algorithm in machine learning, where it often appears in the form of stochastic gradient method (SGD). We shall not elaborate on SGD here.

The gradient descent method for solving (8.5)

```
Initialize $x^{(0)}$
Repeat $x^{(k+1)} = x^{(k)} - t^{(k)}\nabla f(x^{(k)})$
Until stopping criteria is satisfied
```

The positive number $t^{(k)}$ is called the step size at kth iteration. It is common to take the same step size throughout all iterations.

Example 8.8. $f(x) = x^2$. We start with $x^{(0)} = 3$ and pick $t^{(k)} = 0.2$ for all iterations. $f'(x) = 2x$. Then we get the iteration formula $x^{(k+1)} = x^{(k)} - 0.4x^{(k)} = 0.6x^{(k)}$.

$$x^{(1)} = 0.6 * 3 = 1.8, \quad x^{(2)} = 0.6 * 1.8 = 1.08, \cdots, x^{(k)} = x^{(0)}(0.6)^k, \cdots$$

This is a geometric sequence converging to 0, at which f attains its minimum.

Example 8.9. $f(x_1, x_2) = \exp(-x_1 - x_2)$. We can verify that this function is convex everywhere (not strictly convex). $\nabla f(x) = \exp(-x_1 - x_2)(-1, -1)$. Pick $t^{(k)} = 1$ and $x^{(0)} = (0, 0)$

$$x^{(1)} = -e^0(-1, -1) = (1, 1), \quad x^{(2)} = (1, 1) - e^{-2}(-1, -1) = (1 + e^{-2}, 1 + e^{-2}), \cdots$$

Remark 8.10. How to pick the step size $t^{(k)}$ is a big research topic. Small values of t gives slow convergence and big values of t may result in divergence. We will partially address this issue in Chapter 10 when we deal with $Ax = b$.

For a general nonconvex function, there are (usually) many local minimums. If we apply the gradient method on a nonconvex function, the iterates are likely to converge too, but to a local minimum. This is partially why it is much easier to solve a convex optimization problem.

8.2.2 Newton's Method

Newton's method in Chapter 1 can be generalized to the multivariable case. Recall that in order to solve $f(x) = 0$, we use the iteration $x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$. Thus if we are solving $f'(x) = 0$, we can iterate as $x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}$.

As mentioned, we are solving $\nabla f = 0$ in an unconstrained convex problem, so the Newton's formula can be adapted to

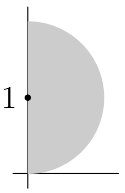
$$x^{(k+1)} = x^{(k)} - t^{(k)} \left(Hf(x^{(k)})\right)^{-1} \nabla f(x^{(k)}).$$

The disadvantage is that we have to solve for a linear system $(Hf(x^{(k)})x = \nabla f(x^{(k)}))$ in every iteration.

Exercises

1. For the problem "find the point on the line $x + y = 1$ that is closest to (-1,0).", reformulate it in the form of (8.1) and determine whether it is a convex optimization problem.

2. Find the gradient of $f(x_1, x_2, x_3) = x_1 e^{x_2} - 2x_3x_2 + \sin(x_1x_3)$.

3. 

Rewrite this convex set (gray area, including boundary) as $\{f_i(x) \leq 0, i = 1, \cdots, m\}$ where f_i are convex functions.

4. In Example 8.8

   (a) What happens if we pick $t^{(k)} = 1$?

   (b) Pick $t^{(k)} = a$. Determine the range of a so that the sequence will converge to 0.

5. In Example 8.9

   (a) Verify the function f is convex everywhere.

   (b) With $x^{(0)} = (0, 0)$, what step size should you pick (different step size in each iterate) in order to have $x^{(k+1)} = x^{(k)} + (1, 1)$?

6. Find the domain on which $f(x, y) = -e^{-x^2-y^2}$ is convex.

Chapter 9

The eigenvalue problem

Recall that $v \neq 0$ is an eigenvector of A if $Av = \lambda v$, where λ is the corresponding eigenvalue. Finding the eigenvalue/eigenvector of a square matrix is a very important problem in numerical linear algebra, and has numerous applications. For instance, solving differential equations that rise from physics and chemical engineering need to find eigenvalues. Principal Component Analysis and spectral graph theory in data analysis is all about eigenvalues.

We reviewed how to find eigenvalues in Section 3.2. That is to find the roots of the characteristic polynomial of A: $\det(A - \lambda I)$. This is the method presented in 99% of linear algebra textbooks, and is taught in a normal linear algebra course. It is important in theory and however an unpractical method for finding eigenvalues. First of all, it is a tremendous amount of work to find this polynomial. To make matters worse, polynomial root finding is an unstable problem, meaning that the roots can change drastically with a tiny change on the coefficients. For example, consider the roots of $x^2 = 0$ and $x^2 + 0.001 = 0$. Finally, as mentioned at the very beginning, there isn't a formula for finding the roots of a polynomial if its degree is higher than 4. The quadratic formula has been known since ancient times. Formula for roots of degree 3 or 4 was found in the 16th century. Mathematicians were looking for a formula for higher degree polynomials for several centuries until Abel (a Norwegian mathematician who died at age of 26) announced the striking news in 1824 that there isn't a formula (party is over, everybody go home now...). Galois, who died even younger at the age of 20, made similar claims using a more sophisticated method, which resulted the start of group theory (consequently modern algebra). No formula is not the end of story for polynomial root finding. In fact, this problem has never been more popular. Algebraic Geometry, perhaps the hottest branch of pure mathematics, is all about studying the roots of multivariate polynomials.

In summary, studying zeros of a polynomial is a fascinating problem, but useless for finding eigenvalues practically. The hope for finding an eigenvalue with a direct solution (aka, a formula) is gone because otherwise there would be a formula for roots of polynomials. Therefore the best we can do is to find an approximate using iterations. We will need to introduce relevant theories first.

9.1 Eigenvalue-revealing decompositions

Using the characteristic polynomial, it is trivial to find the eigenvalues of an upper triangular or lower triangular matrix - the diagonals. What about a general matrix? The goal is to turn it to a triangular matrix by performing eigenvallue-preserving transformations. We assume all matrices are $n \times n$ unless specified otherwise.

Two matrices A and B are called *similar* if there exists an invertible matrix U such that

$$A = UBU^{-1}. \tag{9.1}$$

Similar matrices are matrix representations of the same linear transformations under different bases, therefore similar matrices have the same eigenvalues.

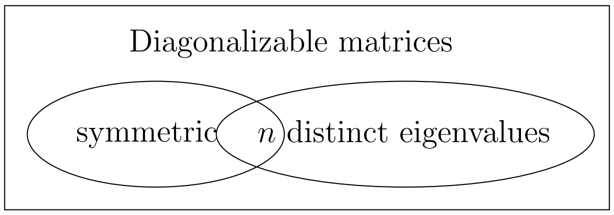
To repeat, if we apply a matrix on the left and its inverse on the right, we are not changing the eigenvalues (corresponding eigenvectors likely change). This is the foundation of all eigenvalue finding algorithms. We will call $A \to WAW^{-1}$ a *similarity transformation.*

A diagonalizable matrix is a matrix that is similar to a diagonal matrix, i.e., $A = UDU^{-1}$ where D is diagonal. This is also called the diagonalization of A, and can be rewritten as $AU = UD$. If we further write $U = [u_1, \cdots, u_n]$ and $D = diag(\lambda_1, \cdots, \lambda_n)$, then

$$A[u_1, \cdots, u_n] = [u_1, \cdots, u_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \Longleftrightarrow Au_i = \lambda_i u_i.$$

This tells us that if A is diagonalizable as $A = UDU^{-1}$, then each column of U is an eigenvector, and the corresponding eigenvalues are stored in the diagonal matrix D. All these eigenvectors are independent because U is invertible. This is why we say in Section 3.2 that if A has n independent eigenvectors, it is diagonalizable.

Not every matrix is diagonalizable. Compute the eigenvectors of $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ for a quick non-diagonalizable example. Obviously a diagonal matrix is diagonalizable ($D = IDI^{-1}$), but there are two kinds of matrices that are for sure diagonalizable: a symmetric matrix or a matrix with n distinct eigenvalues. These two kinds of matrices are useful examples of diagonalizable matrices, but there are other diagonalizable matrices that belong to neither. The following diagram illustrates the relationships:



For a symmetric matrix S, not only it is diagonalizable, it can be diagonalized by an orthonormal matrix Q:

$$S = QDQ^{-1} = QDQ^T. \tag{9.2}$$

In other words, the eigenvectors corresponding to different eigenvalues must be orthogonal to each other. (and the eigenvectors corresponding to the same eigenvalue can be made orthogonal too by Gram-Schmidt.)

For a general matrix, we can do *Schur decomposition*: For any square matrix A, there exists an orthonormal matrix Q and an upper triangular matrix T such that

$$A = QTQ^{-1} = QTQ^T. \tag{9.3}$$

Schur decomposition is an eigenvalue revealing decomposition as the diagonals of T are the eigenvalues of A. Moreover, it says that every matrix is similar to an upper triangular matrix.

9.2 The Hessenberg form

A matrix $B = (b_{ij})$ is called upper tridiagonal or in the Hessenberg form if $b_{ij} = 0$ for $i \geq j+2$. It looks like

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}$$

which is "one line away from upper triangular".

There is not a direct method to find the Schur decomposition because there is not a direct method to find eigenvalues in general. The best we can do is to find B in Hessenberg form that is similar to A. We will use Householder reflector again, as done in the QR decomposition. The process is similar too. Below is a chart of this process for a 5 by 5 matrix.

$$\underset{A_0 = A}{\begin{bmatrix} * & * & * & * & * \\ \times & * & * & * & * \\ \times & * & * & * & * \\ \times & * & * & * & * \\ \times & * & * & * & * \end{bmatrix}} \to \underset{A_1 = Q_1 A_0 Q_1^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & \times & * & * & * \\ & \times & * & * & * \\ & \times & * & * & * \end{bmatrix}} \to \underset{A_2 = Q_2 A_1 Q_2^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & \times & * & * \\ & & \times & * & * \end{bmatrix}} \to \underset{B = Q_3 A_2 Q_3^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}}$$

To construct Q_1, let a_1 be the first column of A with first entry discarded (the vector marked by $\times$), and $u_1 = a_1 + \text{sign}(\text{first coordinate of } a_1)\|a_1\|e_1$ for stability as in Remark 6.13. The the Householder reflector that sends a_1 to a multiple of e_1 is $H_1 = I - 2\dfrac{u_1 u_1^T}{u_1^T u_1}$. Let $Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix}$, then $Q_1 A Q_1^T = \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} \begin{bmatrix} a & b \\ a_1 & C \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & H_1^T \end{bmatrix} = \begin{bmatrix} a & bH_1^T \\ H_1 a_1 & H_1 C H_1^T \end{bmatrix}$. The vector $H_1 a_1$ has all entries 0 except for the first entry.

In general, $Q_k = \begin{bmatrix} I_k & 0 \\ 0 & H_k \end{bmatrix}$ after the construction of H_k.

Example 9.1. Let $A = \begin{bmatrix} 2.5 & -2 & -3 & 4.5 \\ -0.5 & 1 & 3 & -1.5 \\ 1 & 1.5 & -2.5 & 2 \\ 2 & 0.5 & -0.5 & 0 \end{bmatrix}$. Find an orthonormal matrix Q such that QAQ^T is in the Hessenberg form.

$a_1 = (-0.5, 1, 2), u_1 = a_1 - \|a_1\|_2 e_1 = (-2.791, 1, 2)$. $H_1 = I - 2\dfrac{u_1 u_1^T}{u_1^T u_1}$

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.218 & 0.436 & 0.873 \\ 0 & 0.436 & 0.844 & -0.313 \\ 0 & 0.873 & -0.313 & 0.375 \end{bmatrix}, A_1 = Q_1 A Q_1^T = \begin{bmatrix} 2.5 & 3.055 & -4.811 & 0.878 \\ 2.291 & -0.095 & -1.835 & 1.894 \\ 0 & 0.283 & -0.191 & 1.937 \\ 0 & -0.415 & 3.574 & -1.214 \end{bmatrix}$$

$a_2 = (0.283, -0.415), u_2 = a_2 + \|a_2\|_2 e_1 = (-0.220, -0.415)$. $H_2 = I - 2\dfrac{u_2 u_2^T}{u_2^T u_2}$

$$Q_2 = \begin{bmatrix} I_2 & 0 \\ 0 & H_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -0.563 & 0.826 \\ 0 & 0 & 0.826 & 0.563 \end{bmatrix}, B = A_2 = Q_2 A_1 Q_2^T = \begin{bmatrix} 2.5 & 3.055 & 3.436 & -3.480 \\ 2.291 & -0.095 & 2.599 & -0.449 \\ 0 & -0.503 & -3.454 & 1.349 \\ 0 & 0 & -0.288 & 2.049 \end{bmatrix}.$$

$Q = Q_2 Q_1$.

The Hessenberg form B is not eigenvalue-revealing. But it can be viewed as a preprocess before the QR algorithm that will be covered later.

If A is symmetric, then B will be symmetric too, which becomes *tridiagonal*, a matrix that looks like $\begin{bmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & * & * & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}$.

9.3 Power method

We would like to cover a simple method before we introduce the QR algorithm.

If we assume A is diagonalizable and $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$: gap between the biggest eigenvalue and the second biggest eigenvalue in magnitude, then for arbitrary nonzero vector x, $A^k x$ will converge to the eigenvector corresponding to λ_1. To justify, let $Av_i = \lambda_i v_i, i = 1, \cdots, n$. Since A is diagonalizable, $\{v_1, \cdots, v_n\}$ forms a basis of $\mathbb{R}^n$ and $x = \sum_{i=1}^{n} c_i v_i$. Then $A^k x = \sum_{i=1}^{n} c_i A^k v_i = \sum_{i=1}^{n} c_i \lambda_i^k v_i \Longrightarrow \dfrac{1}{\lambda_1^k} A^k x = c_1 v_1 + \sum_{i=2}^{n} c_i \left(\dfrac{\lambda_i}{\lambda_1}\right)^k v_i$. For $i \geq 2$, the ratio $\left(\dfrac{\lambda_i}{\lambda_1}\right)^k \to 0$ as $k \to \infty$, so

$$\lim_{k \to 0} \frac{1}{\lambda_1^k} A^k x = c_1 v_1.$$

Moreover, the error $\left\| \dfrac{1}{\lambda_1^k} A^k x - c_1 v_1 \right\| \leq \sum_{i=2}^{n} |c_i| \left| \dfrac{\lambda_i}{\lambda_1} \right|^k \|v_i\|$ is dominated by the largest term $|c_2| \|v_2\| \left| \dfrac{\lambda_2}{\lambda_1} \right|^k$, which is converging to 0 at a linear rate $\left| \dfrac{\lambda_2}{\lambda_1} \right|$. A closer to 0 linear rate means faster convergence.

In power method iteration, we often normalize the vector to keep it from being too large or small: $v^{(k+1)} = \dfrac{Av^{(k)}}{\|Av^{(k)}\|}$. Once an eigenvector (or an approximate) v is found, the associated eigenvalue can be computed as $\lambda = \dfrac{v^T A v}{v^T v}$. The quotient $\dfrac{v^T A v}{v^T v}$ is called the *Rayleigh quotient* of a vector v.

The Power Iteration

Initialize $v^{(0)}$ such that $\|v^{(0)}\|_2 = 1$	
$v^{(k)} = \dfrac{Av^{(k-1)}}{\|Av^{(k-1)}\|}$	apply A
$\lambda^{(k)} = (v^{(k)})^T Av^{(k)}$	Rayleigh quotient

The power method is not directly used in practice because it requires a big gap between $|\lambda_1|$ and $|\lambda_2|$ for faster convergence and it only finds the biggest eigenvalue/vector pair. However, its theory is useful for developing the practical algorithms.

We list a simple fact that is useful to speed up the power method: If A has eigenvalues λ_i, then $A - \mu I$ has eigenvalues $\lambda_i - \mu$. Furthermore, $(A - \mu I)^{-1}$ has eigenvalues $(\lambda_i - \mu)^{-1}$.

Example 9.2. Suppose A is 5 by 5 and has eigenvalues 10, -9, 8, 7, 6. Then the power method will find the eigenvalue 10 with a linear rate $9/10 = 0.9$.

The eigenvalues of $A + 2I$ is 12, -7, 10, 9, 8. Then the power method applying to $A + 2I$ will find the eigenvalue 12 with a linear rate $10/12 \approx 0.83$. This is a better convergence rate than 0.9. In the end, we can subtract 2 from 12 to get back the eigenvalue of A.

Example 9.3. Suppose A is 5 by 5 and has eigenvalues 10, -9, 8, 7, 6. Then A^{-1} has eigenvalues $\frac{1}{10}, -\frac{1}{9}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}$. The power method applying to A^{-1} will find $\frac{1}{6}$, biggest eigenvalue of A^{-1} with a linear rate $\frac{1}{7} \div \frac{1}{6} = 6/7 \approx 0.86$. $\frac{1}{6}$ corresponds to the smallest (again in magnitude) eigenvalue of A. This means that with inversion, the power method can also find the smallest eigenvalue of A.

What about the eigenvalues in the middle? We can combine the techniques in Example 9.2 and 9.3, and apply power method to $(A - \mu I)^{-1}$. This is called *Inversion Iteration with Shift.* If we want to find the eigenvalue λ_i, we can pick μ to be close to λ_i.

Example 9.4. Suppose A is 5 by 5 and has eigenvalues 10, -9, 8, 7, 6. Let $\mu = 7.9$, which is very close to the eigenvalue 8. Then the eigenvalues of $(A - 7.9I)^{-1}$ are $\frac{1}{2.1}, \frac{1}{-18.8}, \frac{1}{0.1}, \frac{1}{-0.9}, \frac{1}{-1.9}$. Power method on $(A - 7.9I)^{-1}$ will converge to $1/0.1$ (corresponding to eigenvalue 8) at the linear rate $1/9$. This is a very good rate, and it is not hard to see that the closer μ is to 8, the better the convergence rate is.

Remark 9.5. In inversion iteration with shift, the essential line of code is to compute $(A - \mu I)^{-1}v^{(k)}$. But this doesn't mean that we are computing the inverse matrix (never a good idea in numerical linear algebra). This should be interpreted as solving $(A - \mu I)x = v^{(k)}$. Moreover, since we will use use this coefficient matrix $(A - \mu I)$ again and again, we need to do its LU factorization first.

9.4 The (Shifted) QR algorithm

The Unshifted QR algorithm

Initialize $A^{(0)} = A$	
$Q^{(k)}R^{(k)} = A^{(k-1)}$	QR factorization
$A^{(k)} = R^{(k)}Q^{(k)}$	Recombine factors in reverse order
	$A^{(k)}$ is approximating T in the Schur decomposition

Analysis: $A^{(k)} = R^{(k)}Q^{(k)} = [Q^{(k)}]^T Q^{(k)} R^{(k)} Q^{(k)} = [Q^{(k)}]^T A^{(k-1)} Q^{(k)}$. This means that in each iteration, at least the eigenvalues are not being changed. In fact, it can be shown that the QR algorithm above is equivalent to the power iterations and inverse iterations. See Lecture 28 of [2] for more details. We will also state the following theorem without proof.

Theorem 9.6. *Suppose the eigenvalues of A satisfy $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, then the matrices $A^{(k)}$ from the unshifted QR algorithm will converge to T in the Schur decomposition. As a result, the diagonal entries of $A^{(k)}$ converge to engenvalues of A.*

Let $A = \begin{bmatrix} 2.5 & -2 & -3 & 4.5 \\ -0.5 & 1 & 3 & -1.5 \\ 1 & 1.5 & -2.5 & 2 \\ 2 & 0.5 & -0.5 & 0 \end{bmatrix}$, which is the same matrix in Example 9.1. A has eigenvalues -3, -2, 4, and 2. The -2 and 2 are the same in magnitude, which will be a problem for the QR algorithm. If we run the unshifted QR algorithm for 200 times, then we get

$$A^{(200)} = \begin{bmatrix} 4.000 & -3.286 & 4.347 & 1.464 \\ 0.000 & -3.000 & 1.088 & 0.839 \\ 0.000 & 0.000 & 0.937 & -1.080 \\ 0.000 & 0.000 & 2.890 & -0.937 \end{bmatrix}. \tag{9.4}$$

Even worse, if we keep running more iterations, $A^{(k)}$ is not getting further updated.

To overcome this barrier, we use the shifted QR algorithm:

The Shifted QR algorithm

```
Initialize $A^{(0)} = A$
Pick a shift $\mu^{(k)}$                e.g., $\mu^{(k)} = A^{(k-1)}_{nn}$
$Q^{(k)}R^{(k)} = A^{(k-1)} - \mu^{(k)}I$
$A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I$
                              $A^{(k)}$ is approximating $T$ in the Schur decomposition
```

$A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I = [Q^{(k)}]^T Q^{(k)} R^{(k)} Q^{(k)} + \mu^{(k)}I = [Q^{(k)}]^T (R^{(k)}Q^{(k)} + \mu^{(k)}I) Q^{(k)} = [Q^{(k)}]^T A^{(k-1)} Q^{(k)}$.

This strategy works in a similar way that shifted inversion iteration can find all eigenvalues with better convergence. In fact, introducing shifts corresponds to shifts in simultaneous iteration and shifts in inverse iteration processes. See Lecture 29 of [2] for more details.

9.4.1 How to find eigenvectors?

Once an eigenvalue is found, the standard way to compute the corresponding eigenvector is to use inversion iteration. As mentioned earlier, this is the power iteration with A replaced by $(A - \mu I)^{-1}$.

Inverse Iteration for finding eigenvectors

```
Initialize $v^{(0)}$ such that $\|v^{(0)}\|_2 = 1$
Solve $(A - \mu I)w = v^{(k-1)}$          apply $(A - \mu I)^{-1}$
$v^{(k)} = \frac{w}{\|w\|_2}$                       normalize
```

9.4.2 Why the Hessenberg form?

The Shifted QR algorithm converges slow and is too computationally expensive to be practical for a general matrix A. In each iteration, each step (there are 2 main steps) costs $O(n^3)$ flops. But if we have a Hessenberg matrix B, then each step costs only $O(n^2)$ (details omitted). This is a huge saving, especially when n is huge or we want to perform many iterations of Shifted QR to get more accurate eigenvalues.

If A is symmetric, then its Hessenberg form after similarity transforms is tridiagonal as mentioned at the end of Section 9.2. In this case, the savings is even more: each step of the shifted QR algorithm only requires $O(n)$ flops. This is great news because a lot of the practical problems end up finding eigenvalues of a symmetric matrix. For example, the Laplacian matrix in spectral clustering.

9.4.3 Deflation

Thus we assume that the input of the Shifted QR algorithm is always in the Hessenberg form. During an iteration, if some $(i+1, i)$ entry in the Hessenberg matrix is 0, then the problem can be split into two separate problems. For example, $A^{(200)}$ matrix in (9.4) happens to be Hessenberg. The (3,2) entry is 0, which means that eigenvalues of $A^{(200)}$ is

$$(\text{eigenvalues of } \begin{bmatrix} 4.000 & -3.286 \\ 0.000 & -3.000 \end{bmatrix}) \cup (\text{eigenvalues of } \begin{bmatrix} 0.937 & -1.080 \\ 2.890 & -0.937 \end{bmatrix}).$$

9.4.4 Summary of the QR algorithm

There are two phases.

Phase I: Use Householder reflectors to perform similarity transformations on A to turn it to the Hessenberg form B.

Phase II: Use the Shifted QR algorithm on B. Deflate when possible during iterations.

Exercises

1. If A and B are similar, show that they have the same eigenvalues.
2. Given $A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 2 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 2 \\ 0 & 1 & -1 \end{bmatrix}^{-1}$, state the eigenvalues and corresponding eigenvectors of A without any computation.
3. Construct a 3×3 matrix that is not symmetric, has eigenvalues 1,1 and 2, but diagonalizable. Please justify.
4. If Q and P are orthonormal matrices, prove that QP is still orthonormal.
5. Let $A = \begin{bmatrix} 1 & 3 & 4 \\ 3 & 1 & 0 \\ 4 & 0 & 1 \end{bmatrix}$. Find the tridiagonal form of A with similarity transformations. (This is similar to Example 9.1.)

6. Verify that if v is an eigenvector of A, then the Rayleigh quotient $\dfrac{v^T Av}{v^T v}$ is the corresponding eigenvalue.

7. Suppose A has eigenvalues 1, 2, 3, 4.

   (a) Which eigenvalue will the power method (on A) find? What is the convergence rate?

   (b) Which eigenvalue will the power method on $A - I$ find? What is the convergence rate?

   (c) What is the value of a so that the power method on $A - aI$ will find the eigenvalue 4-a with the fastest convergence rate.

   (d) Which eigenvalue will the power method on $(A-1.9I)^{-1}$ find? What is the convergence rate?

8. Partition a 4 by 4 matrix A as $A = \begin{bmatrix} B & C \\ 0 & D \end{bmatrix}$, where B, C, D are all 2×2 matrices. Prove that $\det(A - \lambda I_4) = \det(B - \lambda I_2)\det(D - \lambda I_2)$. (As a consequence, we can deflate: finding the eigenvalues of A is split to finding the eigenvalues of B and finding the eigenvalues of D.)

Chapter 10

Solving system of linear equations: Iterative methods

In this chapter, we aim to solve $Ax = b$, where A is an $n \times n$ invertible matrix and n tends to be very large.

Large linear systems appear in many applications, see Page 327 of [1] for details. The following table shows what are considered large systems, as well as how long it takes to solve such a system for different computers, using Gaussian elimination (or LU factorization). TaihuLight and Jaguar are super computers.

	TaihuLight	Titan/Jaguar	Personal computer
	93 petaflop/s	17.6 petaflop/s	100 gigaflop/s
$n = 10^6$	11 seconds	57 seconds	116 days
$n = 10^7$	3 hours	15.8 hours	317 years
$n = 10^8$	124 days	1.8 year	

When the system is too large, we have to go for iterative methods since Gaussian elimination is the best direct method. As the name suggests, an iterative method produces a sequence of approximates $x^{(1)}, x^{(2)}, \cdots, x^{(k)}, \cdots$, which are hopefully approximating the true solution x better and better. One obvious advantage of iterative methods is easy interruption and restart, meaning that we can always stop at certain iterate, and resume from that iteration later. Iterative methods are particularly good for sparse matrices.

We first define some common terminologies used throughout this chapter. If $x^{(k)}$ is the kth approximation of x, then $e^{(k)} = x - x^{(k)}$ is called the kth error vector, and $r^{(k)} = b - Ax^{(k)}$ is called the kth residual. Notice that $r^{(k)} = Ae^{(k)}$.

10.1 Jacobi and Gauss-Seidel

Example 10.1. When solving $\begin{bmatrix} 10 & 3 \\ 1 & 20 \end{bmatrix} x = \begin{bmatrix} 80 \\ 100 \end{bmatrix}$, we observe that the diagonal entries are a lot bigger than off-diagonal entries, so we solve $\begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix} x = \begin{bmatrix} 80 \\ 100 \end{bmatrix}$ to get an approximation $x^{(1)} = (8, 5)$. Plugging $x^{(1)}$ back into the system, we get $\begin{bmatrix} 10 & 3 \\ 1 & 20 \end{bmatrix} \begin{bmatrix} 8 \\ 5 \end{bmatrix} = \begin{bmatrix} 95 \\ 108 \end{bmatrix}$. The difference of the right hand sides $b - Ax^{(1)}$ is called the residual. To improve the result, we now need to solve $\begin{bmatrix} 10 & 3 \\ 1 & 20 \end{bmatrix} e = \begin{bmatrix} -15 \\ -8 \end{bmatrix}$ and add e to $x^{(1)}$. But we are going to use the same trick as earlier: solving $\begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix} e = \begin{bmatrix} -15 \\ -8 \end{bmatrix}$ instead, whose solution is $(-1.5, -0.4)$. A better approximation is $x^{(2)} = x^{(1)} + (-1.5, -0.4) = (6.5, 4.6)$. For the record, the true solution is $x = (6.5990, 4.6701)$, so we are already getting a very good approximation.

Example 10.2. When solving $\begin{bmatrix} 10 & 1 \\ 10 & 10 \end{bmatrix} x = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$, we observe that the lower triangular entries are a lot bigger than the rest, so we solve $\begin{bmatrix} 10 & 0 \\ 10 & 10 \end{bmatrix} x = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$ to get an approximation $x^{(1)} = (1, 1)$. Plugging $x^{(1)}$ back into the system, we get the residual $b - Ax^{(1)} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$. To improve the result, we need to solve $\begin{bmatrix} 10 & 1 \\ 10 & 10 \end{bmatrix} e = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ and add e to $x^{(1)}$. But we solve $\begin{bmatrix} 10 & 0 \\ 10 & 10 \end{bmatrix} e = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ instead, whose solution is $(-0.1, 0.1)$. A better approximation $x^{(2)}$ is obtained by $x^{(1)} + (-0.1, 0.1) = (0.9, 1.1)$.

Example 10.1 and Example 10.2 are Jacobi method and Gauss-Seidel method respectively. We will present them rigorously below.

Recall that the fixed point iteration $x^{(k+1)} = \varphi(x^{(k)})$ from Chapter 1 will converge to the fixed point of φ if the initial guess satisfies certain condition. This algorithm can also be applied to multivariable functions.

To solve $Ax = b$, we can split A as $A = M - N$ where M is invertible, so $Ax = b \iff x = M^{-1}Nx + M^{-1}b$. This means that x is a fixed point of $\Phi(x) = M^{-1}Nx + M^{-1}b$, so we can use the fixed point iteration $x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$. Remember that you never compute the inverse of a matrix in a practical algorithm, so this iteration really means "$x^{(k+1)}$ is the solution of $Mx = Nx^{(k)} + b$". This is the left of Table 10.1.

This algorithm is often written in the format of the right hand side of Table 10.1. This is because $x^{(k+1)} = M^{-1}(Nx^{(k)} + b) = M^{-1}((M - A)x^{(k)} + b) = M^{-1}(Mx^{(k)} + r^{(k)}) = x^{(k)} + M^{-1}r^{(k)}$.

General Iteration for $Ax = b$
Initialize $x^{(0)}$
Compute $Nx^{(k)} + b$
$x^{(k+1)} = M^{-1}(Nx^{(k)} + b)$

$\iff$

General Iteration for $Ax = b$
Initialize $x^{(0)}$
Compute $r^{(k)} = b - Ax^{(k)}$
Compute $M^{-1}r^{(k)}$
$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)}$

Table 10.1: General Iteration using fixed point method

Different ways of splitting A result in different algorithms. Jacobi method is $A = D - N$, where D is the diagonal entries of A. This is a very reasonable approach since it only takes n flops to solve for $Dx = b$ for a diagonal matrix D. Gauss-Seidel method is $A = L - N$, where L is the lower triangular part of A.

Jacobi Iteration for $Ax = b$	
Initialize $x^{(0)}$	
Compute $r^{(k)} = b - Ax^{(k)}$	$O(n^2)$
Compute $D^{-1}r^{(k)}$	$O(n)$
$x^{(k+1)} = x^{(k)} + D^{-1}r^{(k)}$	$O(n)$

Gauss-Seidel for $Ax = b$	
Initialize $x^{(0)}$	
Compute $r^{(k)} = b - Ax^{(k)}$	$O(n^2)$
Compute $L^{-1}r^{(k)}$	$O(n^2)$
$x^{(k+1)} = x^{(k)} + L^{-1}r^{(k)}$	$O(n)$

It is easy to verify that Example 10.1 is the Jacobi method with $x^{(0)} = (0, 0)$ and Example 10.2 is the Gauss-Seidel iteration with $x^{(0)} = (0, 0)$ as well.

Example 10.3. Solve $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ with Jacobi iteration. $x^{(0)} = (0, 0)$.

$D = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$.
$r^{(0)} = b - Ax^{(0)} = (1, 1)$.
$x^{(1)} = x^{(0)} + D^{-1}r^{(0)} = (0, 0) + (0.5, 0.5) = (0.5, 0.5)$
$r^{(1)} = b - Ax^{(1)} = (1, 1) - (0.5, 0.5) = (0.5, 0.5)$.
$x^{(2)} = x^{(1)} + D^{-1}r^{(1)} = (0.5, 0.5) + (0.25, 0.25) = (0.75, 0.75)$

Example 10.4. Solve $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ with Gauss-Seidel. $x^{(0)} = (0, 0)$.

$L = \begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix}$.
$r^{(0)} = b - Ax^{(0)} = (1, 1)$.
$x^{(1)} = x^{(0)} + L^{-1}r^{(0)} = (0, 0) + (0.5, 0.5) = (0.5, 0.75)$
$r^{(1)} = b - Ax^{(1)} = (0.75, 0)$.
$x^{(2)} = x^{(1)} + L^{-1}r^{(1)} = (7/8, 15/16)$

The true solution of $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is $(1, 1)$, so Gauss-Seidel is converging faster in each iteration. This doesn't mean Gauss-Seidel is a better method than Jacobi. For each iteration, Gauss-Seidel will take longer because there are two steps that costs $O(n^2)$ flops, whereas there is only one such step in Jacobi's iteration. Given the same amount of time, more iterations of Jacobi can be run, but each iteration of Gauss-Seidel does more. It is not clear who wins the competition at this moment.

We first define another quantity related to eigenvalues.

Definition 10.5. The spectral radius of a matrix G is defined as

$$\rho(G) := \max\{|\lambda| : \lambda \text{ is an eigenvalue}\}.$$

Theorem 10.6. *If $\rho(I - M^{-1}A) < 1$, then for the General iteration algorithm described in Table 10.1, we have*

$$\lim_{k\to\infty} \|e^{(k)}\| = 0.$$

The full proof can be found on page 345 of [1]. We will present the idea here.

$$\begin{aligned} &x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)} \\ \Rightarrow &x^{(k+1)} - x = x^{(k)} - x + M^{-1}r^{(k)} \\ \Rightarrow &e^{(k+1)} = e^{(k)} - M^{-1}r^{(k)} = e^{(k)} - M^{-1}Ae^{(k)} = (I - M^{-1}A)e^{(k)} \end{aligned}$$

So the behavior of the matrix $I - M^{-1}A$ affects the convergence rate directly. In general, the smaller $\rho(I - M^{-1}A)$ is, the faster it converges.

Example 10.7. Let $A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$.

$B_J = I - D^{-1}A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix}$, whose eigenvalues are 0.5, -0.5. $\rho(B_J) = 0.5$.

$B_{GS} = I - L^{-1}A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 0 & 1/4 \end{bmatrix}$, whose eigenvalues are 0, 1/4. $\rho(B_{GS}) = 0.25$.

$\rho(B_{GS}) < \rho(B_J)$, which explains, more rigorously, that each iteration of Gauss-Seidel converges better than each iteration of Jacobi for this particular A.

Jacobi method has fast convergence if the diagonal entries are dominating (way bigger than the rest in absolute value), as seen in Example 10.1. This can indeed be converted to $\rho(I - D^{-1}A) < 1$. See [1, Theorem 12.2.5] and Exercise 5 for an example.

Gauss-Seidel has fast convergence if the lower triangular entries are dominating, as seen in Example 10.2.

10.2 More on Positive definite matrices

This section is some prerequisites for the steepest gradient descent and conjugate gradient descent.

Given a positive definite matrix A, we can define the *A-norm* of a vector as

$$\|x\|_A := \sqrt{x^T A x}.$$

The positive-definiteness guarantees the non-negativity of $x^T A x$ so that we can take its square root. We further define two vectors x, y being *A-orthogonal* if $x^T A y = 0$.

Example 10.8. If $A = aI_n$, a positive scalar multiple of the identity matrix, then $\|x\|_A = \sqrt{x^T (aI_n) x} = \sqrt{a}\|x\|_2$. In particular, if A is the identity matrix, then A-norm is the same as the Euclidean norm.

Example 10.9. $A = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix}, x = (1, -2), y = (2, 2)$

$\|x\|_A^2 = x_1^2 + x_2^2 + x_1 x_2 = 1 + 4 - 2 = 3$.

x and y are A-orthogonal because $x^T A y = x_1 y_1 + x_2 y_2 + x_1 y_2 = 1 * 2 + (-2)2 + 1 * 2 = 0$

Positive definite matrices correspond to ellipses. Consider the matrix $P = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$ where $b > a > 0$, then $x^T P x = 1 \Leftrightarrow \dfrac{x_1^2}{1/a} + \dfrac{x_2^2}{1/b} = 1$. This is an ellipse with major axis on the x_1-axis. In fact, (1,0), the direction of x_1-axis is an eigenvector of P with corresponding eigenvalue a. This turns out to be a general rule:

Theorem 10.10. *Given a positive definite matrix P, the equation $x^TPx = 1$ defines an ellipse (or high-dimensional ellipsoid when $n \geq 3$). If P is a 2-by-2 matrix, assume that $\lambda_1 \geq \lambda_2 > 0$ with corresponding eigenvector v_1, v_2, then*

– the major axis lies on v_2=eigenvector of smaller eigenvalue. the major axis has half width $\sqrt{\dfrac{1}{\lambda_2}}$.

– the minor axis lies on v_1=eigenvector of smaller eigenvalue. the minor axis has half width $\sqrt{\dfrac{1}{\lambda_1}}$.

Example 10.11. $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$. It can be computed that $\lambda_1 = 7, v_1 = (1, 2)$ and $\lambda_2 = 2, v_2 = (2, -1)$. As shown in Figure 10.1, red line is the major axis, which has the direction (2,-1), and blue line is the minor axis, which has the direction (1,2). Length of half of red line $= \sqrt{1/2}$. Length of half of blue line $= \sqrt{1/7}$.

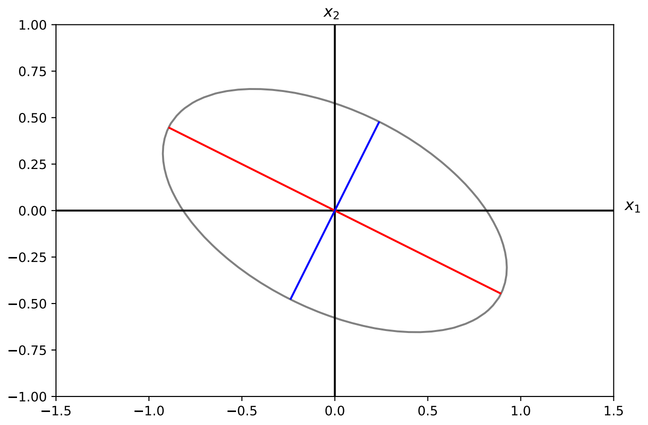


Figure 10.1: Ellipse $x^TAx = 1$ from Example 10.11

We said that a positive definite matrix is related to an ellipse, and in fact many ellipses if we let a vary in $x^TAx = a$. Ellipses can be thought of a stretch from a circle. Two orthogonal vectors stretched the same way that a circle stretched to an ellipse (related to A), will become A-orthogonal. See the visualization in Figure 10.2.

10.3 Steepest Gradient Descent

If A is positive definite, then i $f(x) = \dfrac{1}{2}x^TAx - b^Tx + c$ is a quadratic form. We have discussed in Chapter 8 that this is a strictly convex function, so it is minimized when $\nabla f(x) = 0$.

$$\nabla f(x) = Ax - b,$$

therefore

$$\min_x \frac{1}{2}x^TAx - b^Tx + c \Longleftrightarrow \text{ solve } Ax = b$$

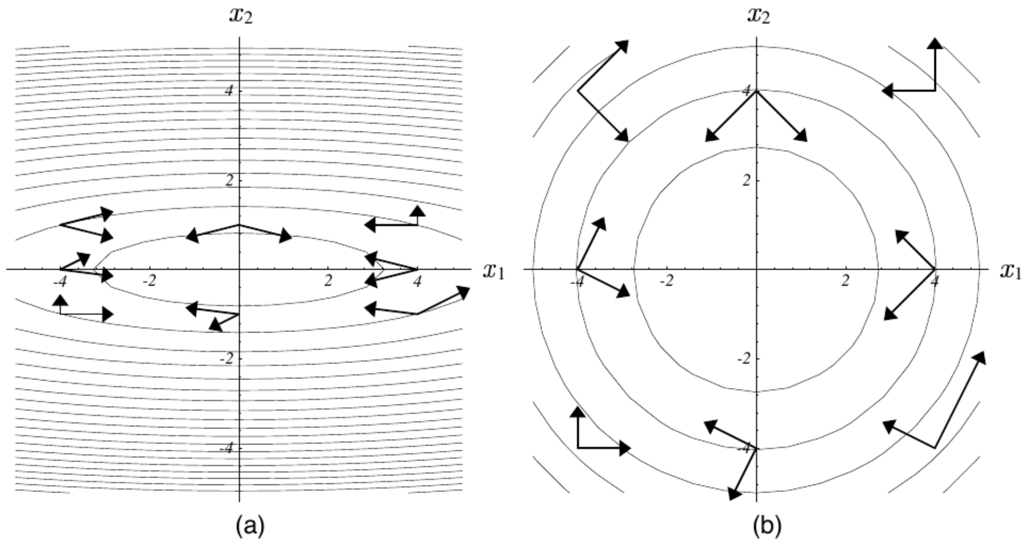


Figure 10.2: A-orthogonal

We have turned the linear system question to a convex optimization, for which we will use the gradient descent method. This formula is useful and will be used a lot:

$$\nabla f(x^{(k)}) = Ax^{(k)} - b = -r^{(k)}.$$

The iteration formula is

$$x^{(k+1)} = x^{(k)} - t^{(k)}\nabla f(x^{(k)}) = x^{(k)} + t^{(k)} r^{(k)}. \tag{10.1}$$

Recall that for a general convex function, it is HARD to figure out the optimal step size $t^{(k)}$ at each step. We have a very specific function here: the quadratic form. We will figure out the best step size using calculus. This is called the *Steepest Gradient Descent* method.

With $x^{(k)}$ and $r^{(k)}$ fixed, we want to find the best step size $t = t^{(k)}$ so that $f(x^{(k+1)})$ is smallest. Let $h(t) = f(x^{(k+1)}) = f(x^{(k)} + tr^{(k)})$.

$$0 = \frac{dh}{dt} = (\nabla f(x^{(k+1)}))^T r^{(k)} = (Ax^{(k+1)} - b)^T r^{(k)} = -(r^{(k+1)})^T r^{(k)}, \tag{10.2}$$

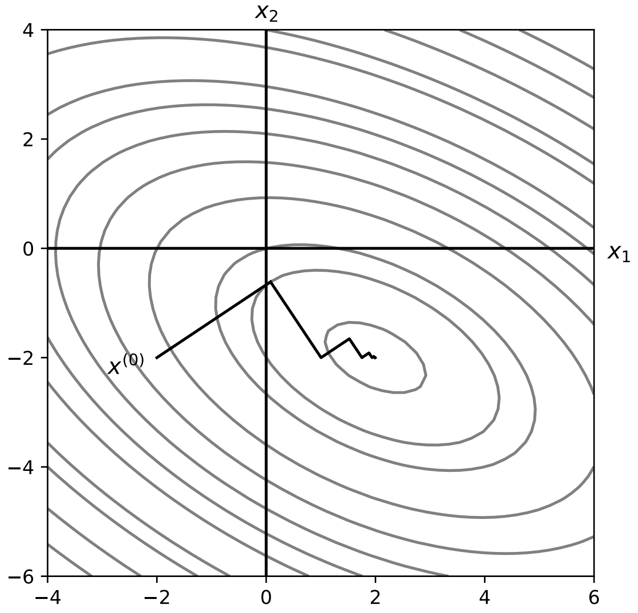
which shows that the next search direction is perpendicular to the previous one, as shown in Figure 10.3.

$$0 = (Ax^{(k+1)} - b)^T r^{(k)} = (Ax^{(k)} + At^{(k)} r^{(k)} - b)^T r^{(k)} = (t^{(k)} Ar^{(k)} - r^{(k)})^T r^{(k)} = t^{(k)} (r^{(k)})^T Ar^{(k)} - (r^{(k)})^T r^{(k)}$$

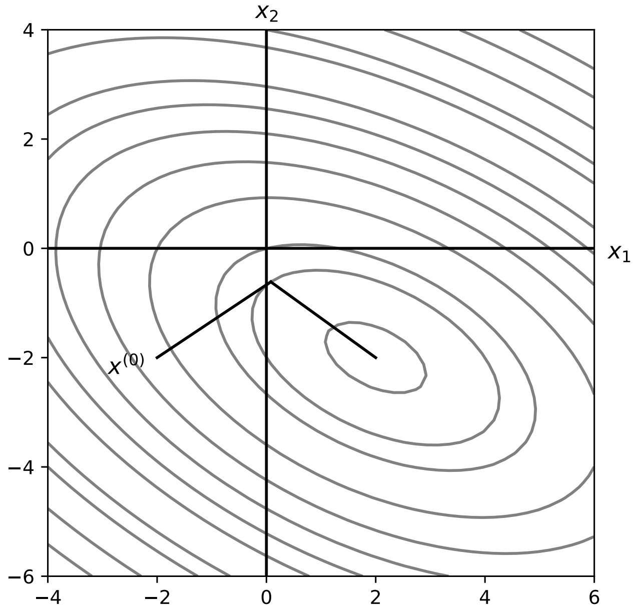
$$\Rightarrow t^{(k)} = \frac{r^{(k)})^T r^{(k)}}{(r^{(k)})^T Ar^{(k)}}. \tag{10.3}$$

Steepest Gradient Descent (A is positive definite)	
Initialize $x^{(0)}$	
Compute $r^{(k)} = b - Ax^{(k)}$	$O(n^2)$
$t^{(k)} = \frac{r^{(k)})^T r^{(k)}}{(r^{(k)})^T Ar^{(k)}}$	$O(n)$
$x^{(k+1)} = x^{(k)} + t^{(k)} r^{(k)}$	$O(n)$

Figure 10.3(a) shows the iterations of steepest gradient descent for solving $\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$ with the initial value $x^{(0)} = (-2, -2)$. Notice the zigzagging towards the true solution $(2, -2)$. This shall be fixed with the Conjugate Gradient Descent algorithm.



(a) Steepest Gradient Descent



(b) Conjugate Gradient Descent

Figure 10.3

10.3.1 Convergence analysis

We need to analyze how good $x^{(k)}$ is approximating x and at what speed. This is equivalent to analyze how small the error vector $e^{(k)} = x - x^{(k)}$ is. It is common to measure the Euclidean norm of $\|e^{(k)}\|_2$. We will use the A-norm because

$$2[f(x^{(k+1)}) - f(x)] = (x^{(k+1)} - x)^T A(x^{(k+1)} - x) = \|e^{(k+1)}\|_A^2.$$

By (10.1), we get the iteration for error vector:

$$e^{(k+1)} = e^{(k)} - t^{(k)} r^{(k)}.$$

Moreover, for ease of writing, we let $e = e^{(k)}, r = r^{(k)}, t = t^{(k)} = \dfrac{r^T r}{r^T A r} = \dfrac{\|r\|_2^2}{\|r\|_A^2}$. So

$$\begin{aligned} \|e^{(k+1)}\|_A^2 &= (e - tr)^T A(e - tr) \\ &= \|e\|_A^2 - te^t Ar - tr^T Ae + t^2\|r\|_A^2 = \|e\|_A^2 - 2tr^T Ae + t^2\|r\|_A^2 \\ &= \|e\|_A^2 - 2tr^T r + t^2\|r\|_A^2 = \|e\|_A^2 - 2\frac{\|r\|_2^2}{\|r\|_A^2}\|r\|_2^2 + \frac{\|r\|_2^4}{\|r\|_A^4}\|r\|_A^2 \\ &= \|e\|_A^2 - \frac{\|r\|_2^4}{\|r\|_A^2} = \|e\|_A^2 \left(1 - \frac{\|r\|_2^4}{\|r\|_A^2\|e\|_A^2}\right) \end{aligned}$$

$w = 1 - \dfrac{\|r\|_2^4}{\|r\|_A^2 \|e\|_A^2}$ affects the convergence rate. The smaller w is, the faster it converges.

We will analyze w if A is 2 by 2, but the general case is similar. For a 2 by 2 matrix A, we assume it has two orthonormal eigenvectors v_1, v_2 with eigenvalues $\lambda_1 \le \lambda_2$. Let $e = c_1 v_1 + c_2 v_2$, then $r = Ae = c_1\lambda_1 v_1 + c_2\lambda_2 v_2$, $Ar = c_1\lambda_1^2 v_1 + c_2\lambda_2^2 v_2$.

$r^T Ar = (c_1\lambda_1 v_1^T + c_2\lambda_2 v_2^T)(c_1\lambda_1^2 v_1 + c_2\lambda_2^2 v_2) = c_1^2\lambda_1^3 + c_2^2\lambda_2^3$ since $\|v_i\|_2 = 1$ and $v_1^T v_2 = 0$.

$e^T Ae = (c_1 v_1^T + c_2 v_2^T)(c_1\lambda_1 v_1 + c_2\lambda_2 v_2) = c_1^2\lambda_1 + c_2^2\lambda_2$.

Then $w = 1 - \dfrac{(c_1^2\lambda_1^2 + c_2^2\lambda_2^2)^2}{(c_1^2\lambda_1^3 + c_2^2\lambda_2^3)(c_1^2\lambda_1 + c_2^2\lambda_2)} = 1 - \dfrac{(c^2+\kappa^2)^2}{(c^2+\kappa^3)(c^2+\kappa)}$, where $c = \dfrac{c_1}{c_2}, \kappa = \dfrac{\lambda_2}{\lambda_1}$ is the condition number of A.

If c is 0, then $w = 0$ (one step convergence, ideal situation); if $\kappa = 1$, then $w = 0$ as well (one step convergence). When solving $Ax = b$, κ is fixed and c is changing throughout the iterations. The worst case in scenario, we still have

$1 - \dfrac{(c^2+\kappa^2)^2}{(c^2+\kappa^3)(c^2+\kappa)} \le \left(\dfrac{\kappa-1}{\kappa+1}\right)^2$, for all c.

So we have the following convergence analysis theorem (true for a general $n \times n$ positive definite matrix).

Theorem 10.12. *With the steepest descent algorithm, the error vector is decreasing as*

$$\|e^{(k+1)}\|_A \le \|e^{(k)}\|_A \cdot \frac{\kappa-1}{\kappa+1}.$$

10.4 Conjugate Gradient Descent

As mentioned, the next search direction is perpendicular to the previous one in steepest gradient descent, which is why there is zigzagging in Figure 10.3 (a). In order to avoid this situation, we actually need to use search directions that are A-orthogonal, that is, we use the iteration formula

$$x^{(k+1)} = x^{(k)} + t^{(k)} p^{(k)}, \tag{10.4}$$

where $p^{(k)}$ is no longer the opposite of the gradient direction, but satisfy $\{p^{(0)}, p^{(1)}, \cdots, p^{(n-1)}\}$ being A-orthogonal.

Suppose $p^{(k)}$ is fixed, then use exactly the same calculation as in (10.2) and (10.3), we get

$$0 = (\nabla f(x^{(k+1)}))^T p^{(k)} = (Ax^{(k+1)} - b)^T r^{(k)} = (Ax^{(k)} + At^{(k)}p^{(k)} - b)^T p^{(k)} = (t^{(k)}Ap^{(k)} - r^{(k)})^T p^{(k)}$$

$$\Rightarrow t^{(k)} = \frac{p^{(k)})^T r^{(k)}}{(p^{(k)})^T A p^{(k)}}.$$

Now we worry about constructing the next search direction $p^{(k+1)}$ so that it will be A-orthogonal to all previous ones. We could use Gram-Schmidt method to achieve this, but it will be too computationally expensive. The whole point of using iterative methods is to be more efficient. Instead, we use

$$p^{(k+1)} = r^{(k+1)} + \alpha^{(k)} p^{(k)}.$$

To ensure $p^{(k+1)}Ap^{(k)} = 0$, $\alpha^{(k)}$ can be computed as $\alpha^{(k)} = -\dfrac{r^{(k+1)})^T A p^{(k)}}{(p^{(k)})^T A p^{(k)}}$. But how to guarantee $p^{(k+1)}$ is also A-orthogonal to $p^{(0)}, \cdots, p^{(k-1)}$? Such analysis is done under a general method that is called the Krylov subspace method, for which we won't get into.

The algorithm is summarized as below.

Conjugatet Gradient Descent (*A* **is positive definite**)	
Initialize $x^{(0)}, r^{(0)} = b - Ax^{(0)}, p^{(0)} = r^{(0)}$	
(i) Compute $Ap^{(k)}$	$O(n^2)$
(ii) $t^{(k)} = \frac{p^{(k)})^T r^{(k)}}{(p^{(k)})^T Ap^{(k)}}$	$O(n)$
(iii) $x^{(k+1)} = x^{(k)} + t^{(k)}p^{(k)}$	$O(n)$
(iv) $r^{(k+1)} = r^{(k)} - t^{(k)}Ap^{(k)}$	$O(n)$
(v) $\alpha^{(k)} = -\frac{r^{(k+1)})^T Ap^{(k)}}{(p^{(k)})^T Ap^{(k)}}$	$O(n)$
(vi) $p^{(k+1)} = r^{(k+1)} + \alpha^{(k)}p^{(k)}$	$O(n)$

Step (i) is added because the vector $Ap^{(k)}$ is being used again and again later. Step (iv) usually is $r^{(k+1)} = b - Ax^{(k+1)}$, but computing $Ax^{(k+1)}$ is expensive for big n so we use the equivalent formula $r^{(k+1)} = r^{(k)} - t^{(k)}Ap^{(k)}$ instead. This is obtained by subtracting x from both sides of (10.4) and then apply A.

Figure 10.3(b) shows the iterations of conjugate gradient descent for solving $\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$ with the initial value $x^{(0)} = (-2, -2)$.

The Conjugate gradient method terminates in n steps theoretically, but usually we don't even run that many iterations. A convergence analysis can be done and the rate is

$$\|e^{(k+1)}\|_A \leq \|e^{(k)}\|_A \cdot \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}.$$

Exercises

1. If we want to write $f(x) = f(x_1, x_2) = x_1^2 + 2x_2^2 + 2x_1x_2 - 2x_1 + 5x_2 + 3$ as the standard quadratic form as $f(x) = \frac{1}{2}x^T Ax - b^T x + c$, what is A, b and c?

2. Given $n \times 1$ vectors x, y and $n \times n$ matrix A, show that $x^T A^T y = y^T Ax$.

3. Given a quadratic form $f(x) = \frac{1}{2}x^T Ax - b^T x + c$. Let x_0 be such that $Ax_0 = b$. Show that $f(y) - f(x_0) = \frac{1}{2}(x_0 - y)^T A(x_0 - y)$. Remember that A is symmetric in a quadratic form. You may find Exercise 2 useful.

4. Let $A = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix}$

   (a) Verify that A is positive definite.

   (b) Compute the A-norm of $x = (1, 1)$.

   (c) Find two vectors that are A-orthogonal.

   (d) Draw the ellipse $x^T Ax = 1$. Indicate the length of major/minor axes.

5. Let $A = \begin{bmatrix} 10 & 3 \\ 1 & 20 \end{bmatrix}$, as in Example 10.1. Compute the spectral radius of $I - D^{-1}A$ to justify the fast convergence in that Example.

6. Let $B = \begin{bmatrix} 10 & 1 \\ 10 & 10 \end{bmatrix}$, as in Example 10.2. Compute the spectral radius of $I - L^{-1}B$ to justify the fast convergence in that Example.

7. Compute the first two iterations $x^{(1)}, x^{(2)}$ of Steepest Gradient Descent applied to the problem $\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$ with the initial value $x^{(0)} = (-2, -2)$. (probably with a calculator)

8. Compute the first two iterations $x^{(1)}, x^{(2)}$ of Conjugate Gradient Descent applied to the problem $\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$ with the initial value $x^{(0)} = (-2, -2)$.

9. Prove that $P^{(1)}$ is A-orthogonal to $p^{(0)}$ in the conjugate gradient descent algorithm.

10. Compute the exact number of flops needed in one iteration of the conjugate gradient descent algorithm.

Bibliography

[1] Anne, Greenbaum, and Timothy P. Chartier. *Numerical methods: design, analysis, and computer implementation of algorithms*. Princeton University Press, 2012.

[2] Lloyd N. Trefethen, and David Bau III. *Numerical linear algebra*. Vol. 50. Siam, 1997.