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

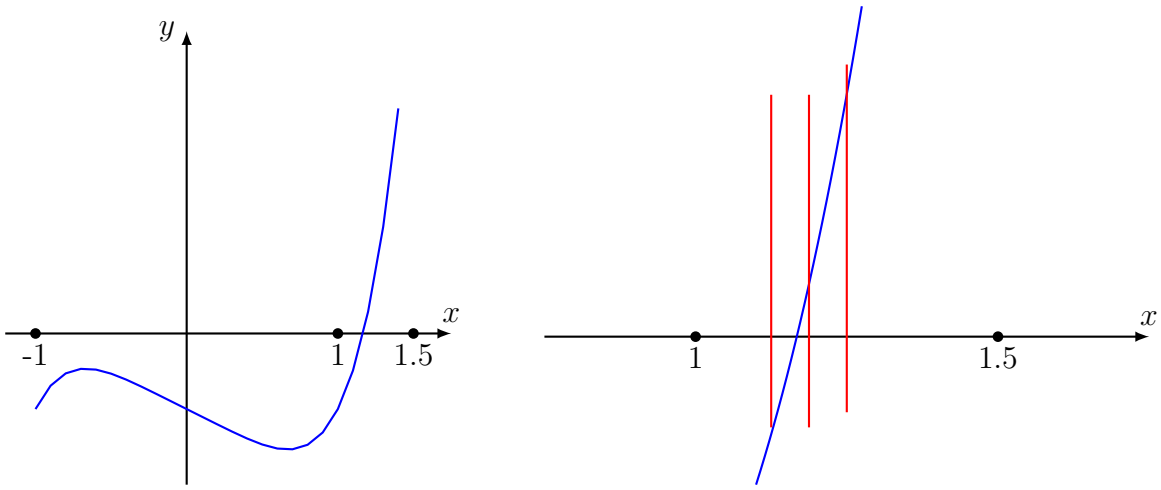


Figure 1.1: Bisection Method

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

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

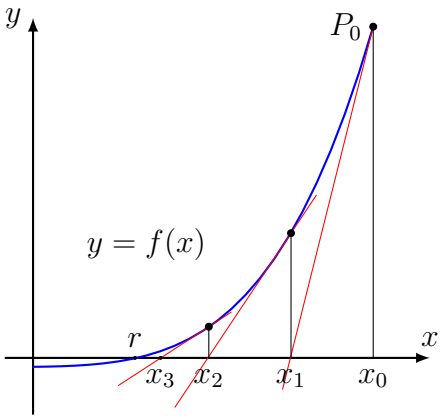


Figure 1.2: Newton's method

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]. □

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 secand 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

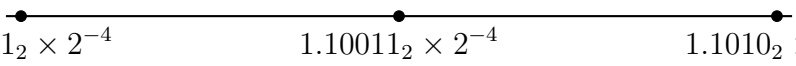
$$0.1 = 0.0\overline{0011}_2$$

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,

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

which is approximately `0.1000000000000000055511151231257827021181583404541015625` in decimal.

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 $|\epsilon| \leq \epsilon_m$, where ϵ_m is the machine precision. This can also be expressed as $\left|\dfrac{fl(x)-x}{x}\right| \leq \epsilon_m$, 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=$\left|\dfrac{1.23-1.2}{1.21}\right| \approx 0.02$

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

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 $y^T x$).

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.

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.

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.

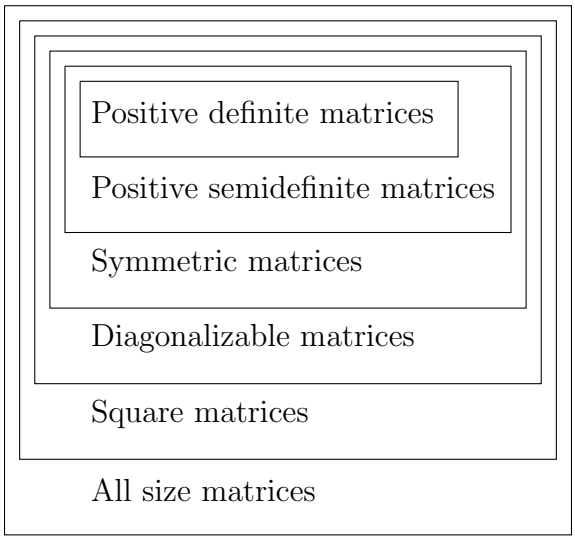


Figure 3.1: Matrices

II. $x^TAx = x_1^2 + 2x_2^2 + 2x_3^2 + 2x_2x_3 = x_1^2 + x_2^2 + x_3^2 + (x_2 + x_3)^2 \geq 0$.
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, $x^TBB^Tx = (B^Tx)^TB^Tx = \|B^Tx\|_2^2 \geq 0$. When equality holds, $B^Tx = 0$ which implies $x = 0$ because B^T is invertible. □

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 $\mathbb{R}^K$:

$$\text{For any } x = (x_1, \cdots, x_K) \in \mathbb{R}^K, \quad x = x_1e_1 + \cdots x_Ke_K.$$

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 $U^TU=I_K$: 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: $x=\sum_{i=1}^{K}c_ib_i$. 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 $x=\sum_{i=1}^{K}c_iu_i$, then*

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

(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 $U^TU=I_K$.

Proof. (a) $u_i^Tx=u_i^T(\sum_{j=1}^{K}c_ju_j)=\boxed{\sum_{j=1}^{K}c_ju_i^Tu_j}=c_iu_i^Tu_i=c_i$. 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) $\|x\|_2^2=\langle x,x\rangle=\langle\sum_{i=1}^{K}c_iu_i,\sum_{j=1}^{K}c_ju_j\rangle=\sum_{i,j}c_ic_j\langle u_i,u_j\rangle=\sum_{i=j}c_i^2=\sum_{i=1}^{K}c_i^2$

(c) This can be computed directly as

$$U^TU=\begin{bmatrix}u_1^T\\ u_2^T\\ \vdots\\ u_K^T\end{bmatrix}[u_1,u_2,\cdots,u_K]=\begin{bmatrix}u_1^Tu_1 & u_1^Tu_2 & \cdots & u_1^Tu_K\\ u_2^Tu_1 & u_2^Tu_2 & \cdots & u_2^Tu_K\\ \vdots & \vdots & \vdots & \vdots\\ u_K^Tu_1 & u_K^Tu_2 & \cdots & u_K^Tu_K\end{bmatrix}=I_K.$$

□

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 $\text{span}\{u_1, \cdots, u_K\}$, see towards the end of Section 4.4.

3.5 Matrix norm

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

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

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) $\|A\| = \sup_{\|x\|=1} \|Ax\|$
(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 $\frac{\|Ax\|}{\|x\|}$ 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 3.11. $\|A\| = \sup_{\|x\|=1} \|Ax\|$

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 3.12. *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 3.13. 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 3.14. $\left\| \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -2 & 2 \end{bmatrix} \right\|_2 = \sqrt{8}$ 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:

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

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 $2n(n-1)+2(n-1)(n-2)+\cdots+0 = \sum_{i=1}^{n} 2i(i-1) = 2\sum_{i=1}^{n}(i^2-i) = \frac{n(n+1)(2n+1)}{3} - n(n+1) \approx \frac{2}{3}n^3$ flops.

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:

$$\left[\begin{array}{cccc:c} 1 & 2 & 2 & \vdots & 1 \\ 2 & 7 & 7 & \vdots & 5 \\ 2 & 7 & 9 & \vdots & 5 \end{array}\right] \xrightarrow[\rho 1(-2)+\rho 3]{\rho 1(-2)+\rho 2} \left[\begin{array}{cccc:c} 1 & 2 & 2 & \vdots & 1 \\ 0 & 3 & 3 & \vdots & 3 \\ 0 & 3 & 5 & \vdots & 3 \end{array}\right] \xrightarrow{\rho 2(-1)+\rho 3} \left[\begin{array}{cccc:c} 1 & 2 & 2 & \vdots & 1 \\ 0 & 3 & 3 & \vdots & 3 \\ 0 & 0 & 2 & \vdots & 0 \end{array}\right].$$

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:

$$\left[\begin{array}{cccc:c} 1 & 2 & 2 & \vdots & 1 \\ 0 & 3 & 3 & \vdots & 3 \\ 0 & 0 & 2 & \vdots & 0 \end{array}\right] \xrightarrow{\rho 3/2} \left[\begin{array}{cccc:c} 1 & 2 & 2 & \vdots & 1 \\ 0 & 3 & 3 & \vdots & 3 \\ 0 & 0 & 1 & \vdots & 0 \end{array}\right] \xrightarrow[\rho 3(-2)+\rho 1]{\rho 3(-3)+\rho 2} \left[\begin{array}{cccc:c} 1 & 2 & 0 & \vdots & 1 \\ 0 & 3 & 0 & \vdots & 3 \\ 0 & 0 & 1 & \vdots & 0 \end{array}\right]$$
$$\xrightarrow{\rho 2/(3)} \left[\begin{array}{cccc:c} 1 & 2 & 0 & \vdots & 1 \\ 0 & 1 & 0 & \vdots & 1 \\ 0 & 0 & 1 & \vdots & 0 \end{array}\right] \xrightarrow{\rho 2(-2)+\rho 1} \left[\begin{array}{cccc:c} 1 & 0 & 0 & \vdots & -1 \\ 0 & 1 & 0 & \vdots & 1 \\ 0 & 0 & 1 & \vdots & 0 \end{array}\right].$$

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 $L_1 A = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 3 & 5 \end{bmatrix}$, where L_1 represents the first two row operations. To figure out L_1, we perform these two row operations on the identity matrix: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow[\rho 1(-2)+\rho 3]{\rho 1(-2)+\rho 2} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} = L_1$. We can figure out $L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$ the same way, and

$$L_2 L_1 A = U,$$

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

So

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

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	$\frac{2}{3}n^3 m$ 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

$x^T y$	$2n - 1 = O(n)$
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

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

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 $x_2 = 1$, $10^{-20}x_1 + 1 = 1 \Longrightarrow x_1 = 0$, which is far from the actual solution.

A remedy of this is to exchange these two rows:

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

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

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

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 $\left[\begin{array}{cccc|c} 2 & -1 & 7 & 3 & 19 \\ 4 & 4 & 0 & 7 & 11 \\ 2 & 1 & 3 & 1 & 9 \\ 6 & 5 & 4 & -17 & -3 \end{array}\right]$.

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.

$$\left[\begin{array}{cccc|c} 0 & -2 & 4 & 2 & 10 \\ 0 & 2 & -6 & 5 & -7 \\ 2 & 1 & 3 & 1 & 9 \\ 0 & 2 & -5 & -20 & -30 \end{array}\right]$$

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.

$$\left[\begin{array}{cccc|c} 0 & -2 & 4 & 2 & 10 \\ 0 & 0 & -2 & 7 & 3 \\ 2 & 1 & 3 & 1 & 9 \\ 0 & 0 & -1 & -18 & -20 \end{array}\right]$$

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 $\boxed{\text{row 2}}$.

$$\left[\begin{array}{cccc:c} 0 & -2 & 4 & 2 & 10 \\ 0 & 0 & -2 & 7 & 3 \\ 2 & 1 & 3 & 1 & 9 \\ 0 & 0 & 0 & -43/2 & -43/2 \end{array}\right] \tag{4.3}$$

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

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

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 $L^T x = y$.	$O(n^2)$

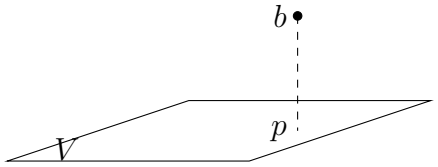


Figure 4.1: Projection onto V

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

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

is called the Projection matrix onto V.

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

$$Ax = Pb. \tag{4.8}$$

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)

$$P = Q(Q^TQ)^{-1}Q^T = QQ^T \tag{4.9}$$

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.

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 $\left[\begin{array}{cc:c} 2 & 2 \times 10^{20} & 2 \times 10^{20} \\ 1 & 1 & 2 \end{array}\right]$. 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. Find the projection matrix onto span$\{(1, 1, 1)\}$.

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

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

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

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

$$A = UDU^T = \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} -10 & 0 \\ 0 & 5 \end{bmatrix} \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix}^T \tag{5.4}$$

Step 3: SVD

Equation (5.4) 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 $\text{span}\{u_1, u_2\} = \text{span}\{[1,1,1,1,1]^T, [1,1,-1,-1,0]^T\}$. The rows of A are approximately on $\text{span}\{v_1, v_2\} = \text{span}\{[1,1,0]^T, [1,-1,0]^T\}$.

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 $\text{span}\{u_1\} = \text{span}\{[1,1,1,1,1]^T\}$. The rows of A are approximately on the line $\text{span}\{v_1\} = \text{span}\{[1,1,0]^T\}$.

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 $V = [v_1, \cdots, v_d]$.

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 $Y = E[v_1, \cdots, v_s]$, V, and c. Which has a total of $\boxed{ms + d^2 + d}$. 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 first stick $d - s$ zero columns to the right of Y, written as $[Y, 0]$, which is an $m \times d$ matrix, then compute

$E' = [Y, 0]V^T$, 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

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

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

$$E' = [Y, 0]V^T = \begin{bmatrix} 1.7264 & 0.7694 & 3.1796 \\ 2.4955 & 2.8909 & -3.0255 \\ -0.3474 & -0.2885 & -0.0673 \\ -0.0108 & -0.6730 & 2.8429 \\ -3.8637 & -2.6988 & -2.9297 \end{bmatrix},$$

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 $r \times n$ matrix consisting of first r rows 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!)

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.

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),

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

We are more interested in the relative error, so

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

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. *$\|A^{-1}\|_2$ is the reciprocal of the smallest singular value of A.*

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.

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.

Bibliography

[1] Anne, Greenbaum, and Timothy P. Chartier. *Numerical methods: design, analysis, and computer implementation of algorithms*. Princeton University Press, 2012.