MA 3257 Numerical Methods for Linear and Nonlinear Systems
Lecture Notes

June 29, 2026

Contents

Part I
Foundations of Numerical Methods

1 Lecture I: Introduction

1.1 Fundamentals of Numerical Methods

Q: What are Numerical Methods?
A: They are **algorithms** to **approximate** the solution to problems in **continuous** mathematics.

Definition 1.1 (Algorithms)**.** *An algorithm is a* ***finite*** *sequence of steps or a formal procedure that a computer program implements.*

1. *Computer programs are always finite.*
2. *Approximate*

   Most of the time, the exact solution to a problem is unavailable.
3. *Continuous*

   There is more to the definition of continuity. Here, it means that the solutions to the mathematical problems are ***real numbers****, not integers. Real numbers are in some sense "infinite", as opposed to discrete/finite like the algorithms.*

Example 1.1. Algebraic equations:

1. $x^2 - 1 = 0$ has exact solutions $x = \pm 1$.
2. $x^2 + 2x - 7 = 0$ has exact solutions via the quadratic formula

$$x = \frac{-2 \pm \sqrt{4 - 4\,(1)\,(-7)}}{2} = 1 \pm 2\sqrt{2},$$

   but $\sqrt{2}$ is a real number, not an integer. An **approximation** is needed.
3. $f\,(x) = x^5 + 2x - 7 = 0$ has no easy formula for the roots. We need **rooting-finding** algorithms to find all points $x \in \mathbb{R}$ such that $f\,(x) = 0$.

Example 1.2. Integrals. Find the area under f for $0 \leqslant x \leqslant 1$.

1. Let $f(x) = e^x$.
$$\int_0^1 e^x dx = e - 1.$$
But, e is an irrational number, which requires an **approximation**.

   (a)
$$\lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$$

   (b)
$$e^x = 1 + x + \frac{x^2}{2!} + \cdots$$

2. Let $f(x) = e^{-x^2}$. Then, $\int_0^1 e^{-x^2} dx$ is impossible since e^{-x^2} does not have an antiderivative expressed by an elementary function. Thus, we approximate it by a numerical integration scheme, such as left point rule, right point, or a little better, trapezoidal (among many numerical integration methods).

Example 1.3. Ordinary Differential Equations (ODEs).

1. Linear ODE

   Suppose we have an initial value problem, $y(0) = y_0$, and $\frac{dy}{dx} = ky$. An exact solution is
$$y(x) = y_0 e^{ky}.$$
   However, the computer will still suffer from the approximation of e.

2. Nonlinear ODE
$$\frac{dy}{dx} = \frac{\sin y}{y} + y^2 x = f(x, y(x)), \quad y(x_0) = y_0.$$
   which has no closed form solution. We need a numerical method to find $y(x)$.
$$y(x) = y_0 + \int_{x_0}^{x} f(x, y(x))\, dx$$
   which asks for a good numerical integration method.

Example 1.4. Numerical Differentiation – Approximation of derivatives.
$$f'(x) = \lim_{z\to x} \frac{f(z) - f(x)}{z - x}$$
where the limit is taken by shrinking the distance from an arbitrary point z to the target point x. This "shrinking" can be done when $|z - x|$ is small enough, but not zero.

Example 1.5. Interpolation of Data: given some experimental data, can we cook up a function that pass through all of them? Then, we will use this function to make estimates where data is unavailable.

1.2 Standards of Algorithms

1. Accuracy: for a given amount of time, how big is the error?
2. Speed: for a given accuracy level, which algorithm is faster?
3. Stability: Can the method fail to yield the correct answer?

To understand the accuracy, speed and stability of an algorithm (and then possibly improve them), we study them **mathematically**. This practice is commonly known as **numerical analysis**. Coding up the algorithm and having a computer running it is the **implementation** of the algorithm.

1.3 Calculus Review

1. A function $f(x)$ is continuous at some point $x = a$ if

$$\lim_{x \to a} f(x) = f(a).$$

   This is a deeper statement than it appears. The limit itself includes both sides of the limit. The LHS is a limiting value, while the RHS is the function value – these two values are NOT necessarily the same. They are the same if and only if the function is continuous.

2. Intermediate value theorem. If f is continuous on $[a, b]$ and $f(a) \leqslant k \leqslant f(b)$, then there exists a point $c \in [a, b]$ such that $f(c) = k$.

1.4 Notations

You may also review Notations_and_Prerequisite_Knowledge.pdf posted on Canvas.

Symbol	Meaning
$\mathbb{R}$	real numbers
$\mathbb{R}^n = \mathbb{R} \times \cdots \times \mathbb{R}$	n-tuple of real numbers, $(a_1, a_2, \ldots, a_n)$.
$\mathbb{Z}$	integers
$\mathbb{Z}^+, \mathbb{N}$	positive integers (natural numbers)
$f : X \to Y$	A function that maps from a set X to another set Y
$C^0([a, b])$	Continuous functions on $[a, b]$
$C^1([a, b])$	Functions with continuous 1st derivative on $[a, b]$
$C^k([a, b])$	Functions with continuous derivatives up to order k on $[a, b]$
$C^\infty([a, b])$	Functions with continuous derivatives up to arbitrary order on $[a, b]$

2 Lecture II: Gaps between Machine Numbers and Rounding Errors

This lecture is based on Chapter III Section 12, Subsection 12.4, in Bornemann.

After class reading: Bornemann Chapter I Section 4 on Execution Times and flops counting. Refer to Chapter I Section 3 for basic MATLAB syntax.

2.1 Binary Machine Numbers

Machines are finite, while each real number has infinite length, i.e. $\frac{1}{3}$, $\sqrt{2}$ and π. Therefore, inaccuracies are guaranteed to arise. How then does a computer store a number so that the inaccuracies are somewhat mitigated?

A 64-bit (binary digit) representation, used for a real number x, is called a floating-point representation of x (the IEEE 754 standard since 1985). We will explore the "floating" feature of this representation in the following example. The floating-point representation of x is commonly written as,

$$fl\,(x) = (-1)^s\, 2^{c-1023}\,(1+f)\,,$$

where s is the **sign** of the number; c, the decimal representation of an 11-bit exponent, each bit 0 or 1, is called the **characteristic**; f, the decimal representation of a 52-bit array (each bit 0 or 1), is called the **mantissa**.

Since c is represented by 11 bits, that is,

$$c_1 c_2 \ldots c_{11}, \quad c_i = 0 \,\text{or}\, 1.$$

The decimal number it represents can be written as a sum of nonnegative powers of 2

$$c = \sum_{i=1}^{11} c_i 2^{i-1}$$

The range of decimal numbers goes from 0 to 2047. Note that to represent 2047, the maximal characteristic, $c_i = 1$ for all i, and thus,

$$\sum_{i=1}^{11} 2^{i-1} = \frac{1\,(1-2^{11})}{1-2} = 2^{11} - 1 = 2047$$

(minimal characteristic is then 0 where $c_i = 0$ for all $i = 1, 2, \ldots, 11$).

However, if we ignore the mantissa and simply use $fl\,(x) = (-1)^s\, 2^{c-1023}$, we can only represent powers of 2 (power ranging from -1023 to 1024). This is still too

coarse of a representation, meaning that the gaps between numbers are too large. The **mantissa** therefore is necessary to represent numbers with even smaller magnitude so as to close the gap between numbers. Consider the 52-bit array,

$$f_1 f_2 \cdots f_{51} f_{52}, \quad f_i = 0 \,\text{or}\, 1.$$

The decimal representation of this array then is

$$f = \sum_{j=1}^{52} f_j \left(\frac{1}{2}\right)^j$$

where f_j is the j^{th} entry of the 52-bit (0 or 1).

Example 2.1. Consider the machine number

$$x = \underbrace{0}_{s,\,\text{sign}} \; \underbrace{10000000011}_{c,\,\text{characteristic}} \; \underbrace{1011100100010\overbrace{0\cdots0}^{40\text{ zeros}}}_{f,\,\text{mantissa}}$$

Here, $s = 0$ (positive real number),

$$c = 2^{10} + 2^1 + 2^0 = 1027.$$

Thus, the exponential part is

$$2^{c-1023} = 2^4.$$

Now,

$$f = \left(\frac{1}{2} + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\right)^8 + \left(\frac{1}{2}\right)^{12}\right).$$

Altogether,

$$(-1)^s \, 2^{c-1023} \, (1 + f) = 27.56640625$$

exactly.

The number immediately smaller than x is

$$x_- = \underbrace{0}_{s,\,\text{sign}} \; \underbrace{10000000011}_{c,\,\text{characteristic}} \; \underbrace{1011100100001\overbrace{1\cdots1}^{40\text{ ones}}}_{f,\,\text{mantissa}},$$

by going down from the $1\overbrace{0\cdots0}^{40\text{ zeros}}$ to $0\overbrace{1\cdots1}^{40\text{ ones}}$. The number immediately bigger than x is

$$x_+ = \underbrace{0}_{s,\,\text{sign}} \; \underbrace{10000000011}_{c,\,\text{characteristic}} \; \underbrace{1011100100010\overbrace{0\cdots0}^{39\text{ zeros}}1}_{f,\,\text{mantissa}}.$$

This means, x represents half the numbers between x_- and x plus half the numbers between x and x_+, namely,

$$\frac{x+x_-}{2} \leqslant x \leqslant \frac{x+x_+}{2}$$

where

$$\begin{aligned}\frac{x+x_-}{2} &= 27.566406249999998223643160599749535322189331054687 5,\\ \frac{x+x_+}{2} &= 27.566406250000001776356839400250464677810668945312 5.\end{aligned}$$

This means, the machine number x alone covers **all** real numbers bounded between the above two rationals. In fact, this gap is called the **machine epsilon** for 64-bit binary representation, and is approximately $2^{-52} \approx 2.22 \times 10^{-16}$ – the reason that it is 2^{-52} is by advancing the mantissa one unit to get the next largest/smallest number.

2.2 Decimal Machine Numbers

While storing the numbers in binary is efficient, it is hard to analyze them in their binary representations. We revert back to the more familiar decimal machine number represetation, in the following floating-point form

$$\pm 0.d_1d_2\ldots d_k \times 10^n, \quad 1 \leqslant d_1 \leqslant 9 \text{ and } 0 \leqslant d_i \leqslant 9, \quad i = 2, 3, \ldots, k.$$

We call numbers of this form k-digit **decimal machine numbers**.

Any positive real number within the numerical range of the machine can be normalized to the form

$$y = 0.d_1d_2\ldots d_kd_{k+1}d_{k+2}\ldots \times 10^n$$

though a termination on the index is certainly needed. There are two commons ways.

2.2.1 Chopping

Simply chop off the digits $d_{k+1}d_{k+2}$ and produce the floating-point form

$$fl\,(y) = 0.d_1d_2\ldots d_k \times 10^n.$$

2.2.2 Rounding

We do it in our daily life all the time. The procedure is to add $5 \times 10^{n-(k+1)}$ and then chops the result to obtain a number of the form

$$fl\,(y) = 0.\delta_1\delta_2\ldots\delta_k \times 10^n.$$

For rounding, when $d_{k+1} \geqslant 5$, we add 1 to the previous digit d_k to obtain $fl(y)$; that is, we **round up**. On the other hand, if $d_{k+1} < 5$, we chop off all but the first k digits; that is, we **round down**. When we round down, $\delta_i = d_i$ for each $i = 1, 2 \ldots, k$.

Example 2.2. Determine the five-digit (a) chopping and (b) rounding values of the irrational number π.

Solution 2.1. The infinite decimal expansion of the form $\pi = 3.1415926535\ldots$. In normalized form, we have

$$\pi = 0.31415926535\ldots \times 10^1.$$

1. Five-digit chopping is

$$fl(\pi) = 0.31415 \times 10^1 = 3.1415.$$

2. Five-digit rounding involves the sixth digit here. Note the sixth digit is 5, so we round up. Using the exact procedure laid out in the above subsection, we add $5 \times 10^{n-(k+1)}$ with $n = 1$ and $k = 5$ (since we want five digits).

$$\begin{aligned} \pi + 5 \times 10^{1-(5+1)} &= 0.314159... \times 10^1 + 5 \times 10^{-6} \times 10^1 \\ &= (0.314159... + 0.000005) \times 10^1 \\ &= 0.314164... \times 10^1 \end{aligned}$$

and thus with five-digit chopping now, we have

$$fl(\pi) = 0.31416 \times 10^1 = 3.1416.$$

The error that results from replacing a number with its floating-point form is called **round-off error** regardless of whether the rounding or the chopping method is used.

3 Lecture III: Basic Error Analysis

3.1 Types of Error

Let p be the true value, and suppose we use p^* to approximate p. The **actual error** is $p - p^*$ (note the order). The **absolute error** is $|p - p^*|$ (always nonnegative). The **relative error** is $\frac{|p-p^*|}{|p|}$ (also always nonngative), provided that $p \neq 0$.

Example 3.1. The relative **round-off error** in approximating a real number y satisfies

$$error = \left|\frac{y - fl\,(y)}{y}\right| = \left|1 - \frac{fl\,(y)}{y}\right|.$$

Suppose we perform the k-digit chopping for decimal representation for the number

$$y = 0.d_1d_2 \ldots d_kd_{k+1}d_{k+2} \ldots \times 10^n,$$

then the relative error is

$$\begin{aligned}
\left|\frac{y - fl\,(y)}{y}\right| &= \left|\frac{0.d_1d_2 \ldots d_kd_{k+1}d_{k+2} \ldots \times 10^n - 0.d_1d_2 \ldots d_k \times 10^n}{0.d_1d_2 \ldots d_kd_{k+1}d_{k+2} \ldots \times 10^n}\right| \\
&= \left|\frac{0.\underbrace{0 \ldots 0}_{k \text{ zeros}}d_{k+1}d_{k+2} \ldots \times 10^n}{0.d_1d_2 \ldots d_kd_{k+1}d_{k+2} \ldots \times 10^n}\right| \\
&= \left|\frac{0.d_{k+1}d_{k+2} \ldots \times 10^{n-k}}{0.d_1d_2 \ldots d_kd_{k+1}d_{k+2} \ldots \times 10^n}\right| \\
&= \left|\frac{0.d_{k+1}d_{k+2} \ldots}{0.d_1d_2 \ldots d_kd_{k+1}d_{k+2} \ldots}\right| \times 10^{-k}.
\end{aligned}$$

Since $d_1 \neq 0$, the minimal value of the denominator is 0.1. The numerator is bounded above by 1. Therefore

$$\left|\frac{y - fl\,(y)}{y}\right| \leqslant \frac{1}{0.1} \times 10^{-k} = 10^{-k+1}.$$

3.2 Significant Digits

We have all been asked to give a numerical answer up to certain number of significant digits. What's the proper mathematical definition?

Suppose you have obtain p as your true answer. It is repeated decimal number that is hard to present. Your professor asks you to provide p^* by recording k significant digits of p.

Definition 3.1 (Significant digit)**.** *The number p^* is said to approximate p to k* ***significant digits*** *if k is the largest nonnegative integer for which*

$$\left|\frac{p-p^*}{p}\right| \leqslant 5 \times 10^{-k}$$

Example 3.2 (Sanity check for significant digits)**.** Suppose we are looking at $p = \pi = 3.1415926535....$ Suppose now a certain algorithm yields

$$p^* = 3.141592234$$

as an approximation to π. We claim that p^* approximates p to 7 significant digits. Is this true? Let's compute the relative error.

$$\begin{aligned}\left|\frac{p-p^*}{p}\right| &= \left|\frac{3.1415926535... - 3.141592234}{3.1415926535...}\right| \\ &= \frac{4.19589793... \times 10^{-7}}{3.1415926535...} \\ &= 1.33559579 \times 10^{-7} \\ &\leqslant 5 \times 10^{-7}\end{aligned}$$

Now, is $k = 7$ the largest nonnegative integer that satisfies the inequality? Let's try $k = 8$. We note that certainly $1.33559579 \times 10^{-7} \nleqslant 5 \times 10^{-8}$ (actually bigger than) – and thus, we go with $k = 7$. So, are we matching π with 7 significant digits?

$$\begin{aligned}\pi &= \underline{3.1415926}535... \\ p^* &= \underline{3.1415922}34\end{aligned}$$

yes, indeed!

Question 3.1. Why is the bound 5×10^{-k}? Why 5 but not 6 or 4?

3.3 Finite-Digit Arithmetic

$$\begin{aligned}x \oplus y &= fl\left(fl\left(x\right) + fl\left(y\right)\right) \\ x \ominus y &= fl\left(fl\left(x\right) - fl\left(y\right)\right) \\ x \otimes y &= fl\left(fl\left(x\right) \times fl\left(y\right)\right) \\ x \oslash y &= fl\left(fl\left(x\right) \div fl\left(y\right)\right)\end{aligned}$$

Example 3.3. Let $x = 2/3$ and $y = 4/7$. Compute the four elementary operations in finite-digit arithmetic.

We try addition first. The true answer is $s = x + y = \frac{26}{21}$ while $x = 0.\overline{6}$ and $y = 0.\overline{571428}$. Using finite-digit arithmetic, say, with 5-digit chopping, we have

$$\begin{aligned} s^* = x \oplus y &= fl\left(fl\left(x\right) + fl\left(y\right)\right) \\ &= fl\left(0.66666 \times 10^0 + 0.57142 \times 10^0\right) \\ &= fl\left(1.23808 \times 10^0\right) \\ &= fl\left(0.123808 \times 10^1\right) \\ &= 0.12380 \times 10^1. \end{aligned}$$

Now, how much did we miss?

$$Error_{abs} = |s - s^*| = \left|\frac{26}{21} - 0.12380 \times 10^1\right| = 9.52 \times 10^{-5},$$

and

$$Error_{rel} = \left|\frac{s - s^*}{s}\right| = \left|1 - \frac{0.12380 \times 10^1}{\frac{26}{21}}\right| = 7.69 \times 10^{-5}.$$

3.4 Catastrophic Cancellations

When two almost equal numbers are subtracted from one another, the floating-point representation of the answer may incur large relative error to the true answer, regardless of whether the individual representation of each number is at a high precision. In the following example, we prescribe a simple four-digit arithmetic to approximate the difference of two almost equal numbers.

Example 3.4. Let $p = 0.85655$ and $q = 0.85642$. Consider four-digit (a) chopping and (b) rounding. Find their absolute and relative errors.

The true answer is $r = p - q = 0.00013$.

1. Chopping yields

$$\begin{aligned} r^* = p \ominus q &= fl\left(fl\left(p\right) - fl\left(q\right)\right) \\ &= fl\left(0.8565 - 0.8564\right) \\ &= fl\left(0.0001\right) \\ &= 0.1 \times 10^{-3}. \end{aligned}$$

Thus

$$Error_{abs} = |r - r^*| = \left|0.00013 - 0.1 \times 10^{-3}\right| = 3 \times 10^{-5}$$

which is not bad. However,

$$Error_{rel} = \left|\frac{r - r^*}{r}\right| = \frac{3 \times 10^{-5}}{0.00013} = \frac{3}{13} \approx 0.231 \leqslant 5 \times 10^{-1}$$

which means we are incurring a 23% relative error – something you can't overlook. The last step shows that the approximation and the true answer has one significant digit of accuracy.

2. Rounding yields

$$\begin{aligned} r^* = p \ominus q &= fl\left(fl\left(p\right) - fl\left(q\right)\right) \\ &= fl\left(0.8566 - 0.8564\right) \\ &= fl\left(0.0002\right) \\ &= 0.2 \times 10^{-3}. \end{aligned}$$

Absolute error is still okay,

$$Error_{abs} = \left|r - r^*\right| = \left|0.00013 - 0.2 \times 10^{-3}\right| = 7 \times 10^{-5}.$$

But, relative error is even worse,

$$Error_{rel} = \left|\frac{r - r^*}{r}\right| = \frac{7 \times 10^{-5}}{0.00013} = \frac{7}{13} \approx 0.538 = 53.8\%,$$

more than 50% relative error! The last step shows that the approximation vs true answer have zero significant digits in common! We are now really comparing apples to oranges.

In practice, the use of algebraic formulas to solve certain problems must be taken with care. A canonical example would be the quadratic formula.

$$x_+ = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_- = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Now, suppose $b > 0$ but $b^2 \gg 4ac$ (meaning that b^2 is far larger than $4ac$). An example of a quadratic equation would be

$$x^2 + 10^5 x + 1 = 0.$$

The negative root x_- is in fact okay, since we are doing addition $-\left(b + \sqrt{b^2 - 4ac}\right)$ in the numerator. However, the root x_+ is subject to catastrophic cancellation because

$$\sqrt{b^2 - 4ac} \approx \sqrt{b^2} = b$$

so $\sqrt{b^2 - 4ac}$ and b are pretty close to each other. As a result, we expect x_+ to be pretty small, and sometimes, so small that the rounding error eats up a chunk of it.

Example 3.5. Consider the quadratic equation $x^2 + 75x + 1 = 0$. True answers to a high precision are the following

$$x_+ = -0.01333570454, \quad x_- = -74.9866642955.$$

Let us use four-digit rounding in computing the roots using the quardratic formula.

$$\sqrt{b^2 - 4ac} = \sqrt{75^2 - 4} = \sqrt{5621} = 74.97.$$

Therefore,

$$fl\,(x_+) = \frac{-75 + 74.97}{2.00} = \frac{-0.03}{2} = -0.015.$$

This approximation of the root has relative error

$$\left|\frac{x_+ - fl\,(x_+)}{x_+}\right| = \left|\frac{x_+ - (-0.015)}{x_+}\right| = 0.1248 = 12.48\%$$

about 12.5% relative error. This is awful.

Solution 3.1. So, what would be the remedy to such atrocious relative error? We simply need to turn around the subtraction into addition. Blessed by the conjugate of the square root expression, we can rewrite

$$\begin{aligned} x_+ &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \\ &= \frac{-b + \sqrt{b^2 - 4ac}}{2a}\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}} \\ &= -\frac{1}{2a}\frac{4ac}{b + \sqrt{b^2 - 4ac}} \\ &= \frac{-2c}{b + \sqrt{b^2 - 4ac}} \end{aligned}$$

Now, the denominator involves an innocent addition. In this case, even if we use the same precision as in the last example, we have the following much better error control,

$$\begin{aligned} fl\,(x_+) &= fl_{\text{on all operations}}\left(-\frac{2.000}{75.00 + \sqrt{75^2 - 4}}\right) \\ = -\frac{2.000}{fl\,(75.00 + 74.97)} &= -fl\left(\frac{2.000}{150.0}\right) = -0.01333. \end{aligned}$$

Thus, the relative error is

$$err_{\text{rel}} = \left|\frac{x_+ - fl\,(x_+)}{x_+}\right| = \left|\frac{-0.01333570454 - (-0.01333)}{-0.01333570454}\right| \approx 0.0004277 = 0.04\%$$

Solution 3.2. Another remedy would be utilizing the relationship between the two roots. In a quadratic equation, $x^2 + bx + c = 0$, we know

$$\begin{aligned} x_+ + x_- &= -b \\ x_+ x_- &= c \end{aligned}$$

Using the good root approximation x_-, we rewrite

$$x_+ = \frac{c}{x_-}$$

where we avoided any sort of subtraction.

Takeaway 3.1 (Quadratic Formula Fix-up)**.**

- Do conjugation to avoid catastrophic cancellation in the implementation of x_+ from the quadratic formula.
- At a higher level, analyze the formula with pen-and-paper work and provide a solution to any numerical roundoff issues using relevant math knowledge before you implement it in practice.

3.5 Nested Arithmetic

Evaluating a function is a common step in many numerical algorithms. The most kind of function is polynomials as many more complicated functions in fact use their Taylor expansions as their representations, which are after all polynomials. Polynomials of order k are of the form

$$p_k(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_k x^k.$$

Evaluating powers of a number thus has become particularly important. We don't want to lose significant digits there (or not lose too many).

Example 3.6. Suppose $x = 4.71$. Evaluate the polynomial $p_3(x) = 1.5 + 3.2x - 6.1x^2 + x^3$ at this point.

We first form a table of straightforward computation (naive computation).

	x	x^2	x^3	$6.1x^2$	$3.2x$
exact	4.71	22.1841	104.487111	135.32301	15.072
3-digit chopping	4.71	22.1	104	134	15.0
3-digit rounding	4.71	22.2	105	135	15.1

Firstly, we know $x^2 = 4.71^2 = 22.1841$. This rounds to 22.2 if we use three-digit rounding. Then,

$$\begin{aligned} fl\left(x^3\right) &= x \otimes x \otimes x \\ &= (x \otimes x) \otimes x \\ &= fl\left(fl\left(x\right) \times fl\left(x\right)\right) \otimes x \\ &= fl\left(4.71^2\right) \otimes x \\ &= 22.2 \otimes x \\ &= fl\left(fl\left(22.2\right) \times fl\left(x\right)\right) \\ &= fl\left(22.2 \times 4.71\right) \\ &= fl\left(104.562\right) \\ &= 105 \end{aligned}$$

which then rounds to 105. Similarly, $6.1x^2 = 135.42$ and rounds to 135, and $3.2x = 15.072$ and rounds to 15.1.

The exact result is

$$p_3\left(4.71\right) = -14.263899.$$

With three-digit chopping, we have

$$f\left(4.71\right) = ((104 - 134) + 15.0) + 1.5 = -13.5, \implies Error_{rel} = \left|\frac{-14.263899 + 13.5}{-14.263899}\right| \approx 0.05$$

and with three-digit rounding, we have

$$f\left(4.71\right) = ((105 - 135) + 15.1) + 1.5 = -13.4, \implies Error_{rel} = \left|\frac{-14.263899 + 13.4}{-14.263899}\right| \approx 0.06$$

(Check these!). Both rounding yield considerable relative error.

The root of the problem lies in the number of arithmetic computations performed by naive/direct computation. Let us count the number of floating-point operations (flops, in short). In the polynomial $p_3\left(x\right) = x^3 - 6.1x^2 + 3.2x + 1.5$, we have

	$+/-$	$\times/\div$
x^3	1	2
$6.1x^2$	1	1
$3.2x$	1	1

which totals 7 flops – note in $6.1x^2$, we already know x^2 from computing x^3, so the only multiplication is $6.1 \times x^2$. To reduce the number of operations, we consider a

nested formulation of the polynomial.

$$\begin{aligned} p_3(x) &= x^3 - 6.1x^2 + 3.2x + 1.5 \\ &= \left(x^2 - 6.1x + 3.2\right) x + 1.5 \\ &= \left(\left(x - 6.1\right) x + 3.2\right) x + 1.5 \end{aligned}$$

Now, we still have 3 $+/-$ as before, but the number of multiplication reduces to just two times. With this formulation, we only incur a total of 5 flops (instead of 7). In a much larger computation, the saving will be significant.

Indeed, still using three-digit chopping but now employing the nested polynomial, we have

$$f(4.71) = \left(\left(4.71 - 6.1\right) 4.71 + 3.2\right) 4.71 + 1.5 = -14.2.$$

We observe that we already hit three significant digits. The relative error of this calculation is

$$Error_{rel} = \left| \frac{-14.263899 + 14.2}{-14.263899} \right| \approx 0.0045,$$

far better than direct computation.

Remark 3.1. *Moral of the story: you* ***always*** *put polynomials in nested form before doing* ***any*** *computations/evaluations.*

4 Lecture IV: Conditioning, Stability and Rate of Convergence

4.1 Well-conditioned Problem

In calculus, we have learned the notion of continuity, that is, we say a function $f(x)$ is continuous at some point $x = a$ if and only if the following statement is true: for every $\epsilon > 0$, we can find a $\delta > 0$ such that

$$|f(x) - f(a)| < \epsilon$$

if $|x - a| < \delta$.

However, this is awfully abstract. What it really means is that given an input very close to $x = a$ (closeness measured by δ), then the output won't be also so far away from the true output (measured by ϵ).

This definition helps us establish something call **well-conditioned** problems. Suppose we are asked to solve a system of equations in matrix form, i.e.,

$$Ax = b.$$

We say this problem is **well-conditioned** if for every perturbation δA and δb, the solution $\tilde{x}$ to the perturbed problem

$$(A + \delta A)\,\tilde{x} = (b + \delta b)$$

is not too far off from the true solution x. In other words, this problem doesn't go crazy when nudged. Otherwise, we call the problem **ill-conditioned**. The Bornemann text has a further discussion of how "crazy" a problem becomes to be considered **ill-conditioned**. We will return to this when we talk about the **condition number**.

More generally, one may think of a problem as a mapping from the data space to the solution space, more precisely,

$$f : \mathcal{D} \to \mathcal{S}.$$

In the example of a system of equations, the function is in fact, "2-dimensional",

$$f(A, b) = A^{-1}b = x$$

where we certainly seek the inverse of the matrix. With input data b and the known operations A, we seek a solution $x = A^{-1}b$. If f is "continuous" in both arguments, then we say this system of equation is a well-conditioned problem. This further suggests that the well-conditionedness of f depends on the invertibility of A.

4.2 Stability of Algorithms

Meanwhile, to solve the problem, one may use an algorithm, which is an approximation of the real problem. Though an approximation, an algorithm consumes inputs and produces outputs. We call an algorithm **stable** or **unstable**, if we insert perturbed inputs and see if we obtain minor or major fluctuations in outputs. We sometimes refer to inputs as **initial data**. Of course, not all algorithms are stable given any initial data. If one can find a condition on the initial data such that the algorithm is stable, then we say the algorithm is **conditionally stable**.

A more general way of looking at algorithms is to consider the same two spaces mentioned in the first section, but now, an approximation function

$$\hat{f} : \mathcal{D} \to \mathcal{S}.$$

One way of quantifying the stability of an algorithm is by looking at how errors are changing at different stages of the operations.

Definition 4.1. *Suppose that $E_0 > 0$ denotes an error introduced at some stage in the calculations, and E_n represents the magnitude of the error after n subsequent operations.*

1. *(linear) If $E_n \approx CnE_0$ where C is independent of n, then we say the growth of error is **linear**.*
2. *(exponential) If $E_n \approx C^n E_0$ for some $C > 1$, then the growth of error is **exponential**.*

Example 4.1. (Unstable Sequence) Consider the sequence of numbers (possibly representing an iterative algorithm)

$$p_n = c_1 \left(\frac{1}{3}\right)^n + c_2 3^n$$

which solves the recurrence relation

$$p_n = \frac{10}{3} p_{n-1} - p_{n-2}, \quad n = 2, 3, \ldots.$$

with two degrees of freedom, c_1 and c_2 (think of them as the discrete version of "integration constants" of a second-order ODE). They are determined if we specify p_0 and p_1, that is, the "initial conditions". The way to solve second order recurrence relation like the above is to guess first a solution of the form

$$p_n = c_1 r_1^n + c_2 r_2^n.$$

Plugging these in, we have

$$c_1 r_1^n + c_2 r_2^n = \frac{10}{3}\left(c_1 r_1^{n-1} + c_2 r_2^{n-1}\right) - \left(c_1 r_1^{n-2} + c_2 r_2^{n-2}\right).$$

Collecting similar terms, we have

$$c_1 r_1^{n-2}\left(r_1^2 + \frac{10}{3}r_1 - 1\right) + c_2 r_2^{n-2}\left(r_2^2 + \frac{10}{3}r_2 - 1\right) = 0.$$

This implies that r_1 and r_2 must be the solution to the quadratic equation

$$r^2 + \frac{10}{3}r - 1 = 0.$$

Suppose $p_0 = 1$ and $p_1 = \frac{1}{3}$. Then, a straightforward calculation yields

$$c_1 = 1, \quad c_2 = 0$$

which implies, with the specific initial conditions, $p_n = \left(\frac{1}{3}\right)^n$ is the unique solution.

Now, suppose we have used five-digit rounding to compute the sequence given by $p_n = \left(\frac{1}{3}\right)^n$, that is, we begin with $\widehat{p_0} = 1.0000$ and $\widehat{p_1} = 0.33333$ – this first readily modifies the constants $\widehat{c_1} = 1.0000$ and $\widehat{c_2} = -0.12500 \times 10^{-5}$. Let's check this.

$$\begin{aligned} 1.0000 = \widehat{p_0} &= \widehat{c_1} + \widehat{c_2} \\ 0.33333 = \widehat{p_1} &= \frac{1}{3}\widehat{c_1} + 3\widehat{c_2} \end{aligned}$$

which yields a system of equations for $\widehat{c_1}$ and $\widehat{c_2}$. Using a substitution derived from the first equation, we have

$$\widehat{c_1} = 1.0000 - \widehat{c_2} \implies 0.33333 = (1.0000 - \widehat{c_2})\,\frac{1}{3} + \widehat{c_2}\,(3)$$

and thus

$$\left(3 - \frac{1}{3}\right)\widehat{c_2} = 0.33333 - \frac{1}{3} \implies \widehat{c_2} = \frac{0.33333 - \frac{1}{3}}{3 - \frac{1}{3}} = -0.12499\ldots \times 10^{-5}.$$

The five-digit rounding for $\widehat{c_2}$ is -0.12500×10^{-5}, while $\widehat{c_1} = 1.0000 - \widehat{c_2} = 1.0000 - (-0.00000125) = 1.00000125$ which rounds to 1.0000 (note the leading 1 ate up the decimals).

Thus, the sequence $\widehat{p_n}$ generated by these rounded numbers is

$$\widehat{p_n} = 1.0000\left(\frac{1}{3}\right)^n - 0.12500 \times 10^{-5}\,(3)^n\,,$$

which has round-off error

$$p_n - \widehat{p_n} = \underbrace{0.12500 \times 10^{-5}}_{E_0}\underbrace{(3)^n}_{C^n}.$$

According to our definition of a stable/unstable algorithm, we observe that this procedure is **unstable** because the error grows exponentially with n.

This also informs us that solutions on paper to a problem may not be accurately evaluated by a machine. Finding an analytical formula is nice. But to put it to practice, one must be careful with the algorithm, sometimes, as simple as "plugging numbers in".

Remark 4.1. *Algorithms that give linear errors are stable. Algorithms that give exponential errors are unstable.*

4.3 Condition Number

Remark 4.2. *Back to the condition of a problem. We may compute a* ***condition number*** κ *of a problem to quantify how relative error in inputs may amplify that in outputs. It roughly satisfies the following*

$$\mathcal{E}_{\text{output}} \approx \kappa \mathcal{E}_{\text{input}}.$$

Example 4.2. Consider function evaluation, finding $f(x_0)$ at $x = x_0$ (just like in nested polynomials). We perturb the input by δ. Then, we consider the relative error in output,

$$\begin{aligned}\mathcal{E}_{output} &= \left|\frac{f(x_0+\delta) - f(x_0)}{f(x_0)}\right| = \left|\frac{\delta f'(\xi)}{f(x_0)}\right| = \left|\frac{x_0 f'(\xi)}{f(x_0)}\right|\left|\frac{\delta}{x_0}\right| \\ &\approx \left|\frac{x_0 f'(x_0)}{f(x_0)}\right|\left|\frac{\delta}{x_0}\right| =: \kappa_f(x_0)\,\mathcal{E}_{input}\end{aligned}$$

where $\xi \in (x_0, x_0+\delta)$ (motivated by Mean Value Theorem). The second term in the parenthesis, $\frac{\delta}{x_0}$, is the relative error in input ($\frac{x_0+\delta-x_0}{x_0}$). Thus, we call

$$\kappa_f(x_0) = \left|\frac{x_0 f'(x_0)}{f(x_0)}\right|,$$

the condition number of evaluating $f(x)$ at $x = x_0$. As you can tell, this number can grow pretty large if we have a steep slope (consider evaluating $x = 1$ for $f(x) = \frac{x}{1-x}$).

Remark 4.3. *The "interface" between well-conditioned/ill-conditioned is not that sharp. However, we do find $\kappa < 1$ to represent a well-conditioned problem, and otherwise ill-conditioned.*

Example 4.3. (Well-conditioned problem vs Unstable Algorithm)

Let's revisit function evaluation. Consider $f(x) = \sqrt{1+x} - 1$ and we wish to evaluate numbers $x = x_0 \approx 0$.

$$\kappa_f(x) = \frac{\sqrt{1+x}+1}{2\sqrt{1+x}}$$

so $\kappa_f(0) = 1$, which means it is at least not ill-conditioned (error propagation stays neutral).

Example 4.4. However, to compute $f(x_0)$, we have a simple three-step algorithm:

1. $s_1 = 1 + x_0$ (stable with $\kappa_{s_1}(0) = \frac{x_0 \times 1}{1+x_0} \mid_{x_0=0} = 0$)
2. $s_2 = \sqrt{s_1}$ (stable with $\kappa_{s_2}(s_1) = \frac{s_1 \frac{1}{2\sqrt{s_1}}}{\sqrt{s_1}} = \frac{1}{2}$, regardless of evaluation point)
3. $s_3 = s_2 - 1$ (unstable with $\kappa_{s_3} = \frac{s_2}{s_2 - 1}$ near $s_2 \approx 1$ – consider catastrophic cancellation).

We would say this algorithm is unstable because one of the steps is unstable. The remedy is against to use rationalization to bypass the catastrophic cancellation. The analysis of the algorithm using rationalization is left as a HW exercise (HW2 Problem 1).

From the last example, we see that the most obvious algorithm to solve a well-conditioned problem may not be stable. Here are a few more takeaways:

Takeaway 4.1.

1. Well-/ill-conditioned refers to the problem.
2. Stable/Unstable refers to an algorithm or a numerical process.
3. If the problem is well-conditioned, then there is a stable way to solve it (but not vice versa).
4. If the problem is ill-conditioned, then there is no reliable way to solve it in a stable way.

One must remember that there are always two facets of scientific computing: the problem and the algorithm. You must study both the condition of the problem AND the stability of your algorithm, and then decide whether the problem should be relaxed to improve its condition, or the algorithm should be revamped to improve stability. It is crucial to know both.

4.4 Rates of Convergence

Consider two functions $f(x) = \frac{1}{x}$ and $g(x) = e^{-x}$. Send x to ∞. We know both limits are zero. But, which one does it faster? We actually compare their rates via L'Hôpital,

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{1/x}{e^{-x}} = \lim_{x\to\infty} \frac{e^x}{x} = \lim_{x\to\infty} \frac{e^x}{1} = \infty.$$

So, the quotient blows up, meaning that $f(x)$ is running away faster than $g(x)$ – or in other words, $g(x) \to 0$ faster than $f(x)$ does.

Can we quantify this "rate" when we no longer have differentiability to help us, i.e. bye-bye L'Hôpital?

How about first just a sequence of numbers? For example, compare the rate at which

$$\alpha_n = \frac{1}{n}, \quad \gamma_n = \frac{1}{n^2}$$

converge to 0? We anticipate that γ_n goes to zero faster, in the sense that we can compute

$$\lim_{n\to\infty} \frac{\alpha_n}{\gamma_n} = \lim_{n\to\infty} \frac{1/n}{1/n^2} = \infty$$

which means α_n is much larger than γ_n when n is large. In other words, γ_n is much closer to zero than $1/n$ and thus converges faster! Now, the task is to quantify the rate of convergence of the sequences.

Definition 4.2. *("big O notation") Suppose $\{\beta_n\}_{n=1}^{\infty}$ is a sequence known to converge to zero, and $\{\alpha_n\}_{n=1}^{\infty}$ converges to a number α. If a positive constant K exists with*

$$|\alpha_n - \alpha| \leqslant K\beta_n, \quad \textit{for large } n,$$

*then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with **rate**, or **order**, **of convergence** $O(\beta_n)$ ("big O of β_n"). We also write this as*

$$\alpha_n = \alpha + O(\beta_n), \quad n \to \infty.$$

Though $\{\beta_n\}$ is an arbitrary sequence that goes to zero, we almost always use the form

$$\beta_n = \frac{1}{n^p}, \quad p > 0.$$

We are interested in the largest value of p such that $\alpha_n = \alpha + O(1/n^p)$. For example, if $p = 1$, we say it is 1st-order/linear convergence; $p = 2$, 2nd order/quadratic convergence, and so on.

Example 4.5. $1/n^p = O(1/n^p)$ certainly. How about $\frac{n}{n^2+1}$? It converges to 0 but at which order? One may suspect 1st order because of the power difference. Let's confirm it.

$$\left|\frac{n}{n^2+1} - 0\right| = \frac{n}{n^2+1} \leqslant \frac{n}{n^2} = \frac{1}{n}$$

where the constant $K = 1$, and $\beta_n = \frac{1}{n}$. Yes, indeed, 1st order.

Definition 4.3. *A little more about "big O" notation here: instead of sequences, we may also say certain function is "big O" of some other function, usually to capture the order of growth/decay. The definition in this case is very similar to that for sequences. If we can find a positive real number M and a threshold x_0 such that*

$$|f(x)| \leqslant Mg(x), \quad \forall x \geqslant x_0,$$

then we say that

$$f(x) = O(g(x)), \quad x \to \infty.$$

Example 4.6. Consider $f(x) = 6x^4 - 2x^2 + 5$. We want to capture its behaviour as $x \to \infty$.

$$\left|6x^4 - 2x^2 + 5\right| \leqslant 6x^4 + 2x^2 + 5 \leqslant 6x^4 + 2x^4 + 5x^4 = 13x^4, \quad \forall x \geqslant x_0$$

(in fact, regardless of what x_0 is) where we identified the constant $K = 13$ and $g(x) = x^4$. Thus, we say $f(x) = O(x^4)$ as $x \to \infty$.

Definition 4.4. *Suppose we are no longer interested in sending $x \to \infty$ but rather $x \to a$, then the above condition is modified: if we can furnish a positive real number M and an interval size δ such that*

$$|f(x)| \leqslant Mg(x), \quad 0 < |x - a| < \delta$$

then we say

$$f(x) = O(g(x)), \quad x \to a.$$

Example 4.7. Consider the same function $f(x) = 6x^4 - 2x^2 + 5$, but now we are interested in the asymptotic behaviour as $x \to 1$. Given $\delta > 0$, we look at the points in the neighborhood of $x = 1$, i.e. the points x that satisfy $0 < |x - 1| < \delta$ (need to utilize this bound on x).

Now, these points obviously satisfy

$$-\delta < x - 1 < \delta$$

which is equivalent to

$$1 - \delta < x < 1 + \delta.$$

Thus,

$$\begin{aligned} |f(x)| &= \left|6x^4 - 2x^2 + 5\right| \\ &\leqslant \left|6(1+\delta)^4 + 2(1+\delta)^2 + 5\right| \\ &= 6(1+\delta)^4 + 2(1+\delta)^2 + 5. \end{aligned}$$

The RHS here is just a number. You choose how close you want to be to $x = 1$ (via δ), and you can always bound the function by the constant above. Therefore, identify

$$M = 6(1+\delta)^4 + 2(1+\delta)^2 + 5, \quad g(x) = 1.$$

We say that

$$f(x) = O(1), \quad x \to 1.$$

In practice, we sometimes care about $x \to 0$, or better if we change the variable to $h \to 0$, where h may represent the interval size of a numerical integral, i.e. the step size of a Riemann sum. We wish to approximate some integral using areas of small rectangles spanned by small subintervals. Another example pertains to solutions of ODEs, where we can only take finite but small time steps to approximate solutions to $\frac{d^2y}{dt^2} = -ky$ (Hooke's law).

Definition 4.5. *Suppose that* $\lim_{h\to 0} G(h) = 0$ *and* $\lim_{h\to 0} F(h) = L$. *If a positive constant* K *exists with*

$$|F(h) - L| \leqslant K|G(h)|$$

for sufficiently small h, *then we say*

$$F(h) = L + O(G(h)).$$

Per usual, we care about $G(h) = h^p$ *for* $p > 0$. *We are interested in the largest value of* p *for which* $F(h) = L + O(h^p)$.

Example 4.8. Use the Maclaurin series of $\cos(h)$ up to three terms to show that $\cos(h) + \frac{1}{2}h^2 = 1 + O(h^4)$.

Proof. Let $f(h) = \cos(h)$. Then we need derivatives up to the fourth order since two of them evaluate to zero at $h = 0$.

$$f'(h) = -\sin(h), \quad f''(h) = -\cos(h), \quad f'''(h) = \sin(h), \quad f^{(4)}(h) = \cos(h).$$

and thus about $h = 0$, we have

$$\cos(h) = \cos(0) + hf'(0) + \frac{h^2}{2!}f''(0) + \frac{h^3}{3!}f'''(0) + \frac{h^4}{4!}f^{(4)}(\xi)$$

where $\xi \in (0, h)$ (last step is the error term of Taylor series).

$$\cos(h) = 1 - \frac{h^2}{2!} + \frac{h^4}{4!}\cos(\xi(h))$$

where in the last term, we must write that this ξ depends on h. Moving terms around, we have

$$\cos(h) + \frac{1}{2}h^2 = 1 + \frac{h^4}{4!}\cos(\xi(h)).$$

Furthermore, identify that $F(h) = \cos(h) + \frac{1}{2}h^2$, $L = 1$ (clearly it is the limit of $F(h)$ as $h \to 0$).

$$\left|\cos(h) + \frac{1}{2}h^2 - 1\right| = \left|\frac{h^4}{4!}\cos(\xi(h))\right| \leqslant \frac{1}{24}h^4$$

which we further identify $K = 1/24$ and $G(h) = h^4 \to 0$ as $h \to 0$ (required by definition). Thus, we can say that

$$\cos(h) + \frac{1}{2}h^2 = 1 + O\left(h^4\right), \quad h \to 0.$$

□

Example 4.9. Consider $f(x) = \sin(x)$. We want to capture its behaviour as $x \to 0$. Given $\delta > 0$ such that $0 \leqslant |x| < \delta$, we have

$$\begin{aligned}
\sin(x) &= x - \frac{x^3}{3!}f^{(3)}(\xi) \\
\implies x - \sin x &= \frac{x^3}{3!}f^{(3)}(\xi) \\
\implies |\sin x - x| &= \left|\frac{x^3}{3!}f^{(3)}(\xi)\right| \\
&= \left|\frac{x^3}{3!}(-\cos(\xi))\right| \\
&\leqslant \frac{x^3}{6}
\end{aligned}$$

where $\xi \in (0, \delta)$ (think Mean Value Theorem applied to Taylor polynomials, or Taylor's Theoerem, Section 1.1, Theorem 1.14). We identify $M = 1/6$ and $g(x) = x^3$. In other words,

$$f(x) = \sin(x) = x + O\left(x^3\right), \quad x \to 0.$$

Remark 4.4. *It is very important that one clarifies where the point of interest is when using big oh notation, i.e. it is a characterization of the* ***asymptotic behaviour*** *of $f(x)$, and when something is asymptotic, you state where the independent variable is limiting. More precisely, when one says $f(x) = O(x^2)$, we must specify where x is going. For example, for the same function $f(x) = x^2 + 1$, $f(x) = O(x^2)$ as $x \to \infty$, but $f(x) = O(1)$ as $x \to 0$, because given δ such that $0 < |x| < \delta$, we have*

$$|f(x)| = \left|x^2 + 1\right| \leqslant \delta^2 + 1$$

which is a constant on the RHS (identify $K = \delta^2 + 1$ and $g(x) = 1$), and thus we can say

$$f(x) = O(1), \quad x \to 0.$$

Part II
Direct Methods for Linear Systems

5 Lecture V: Gaussian Elimination

After class reading: Bornemann Chapter II Section 7.

5.1 Preliminaries for Linear Systems

A system of equations

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

can be represented by the matrix equation

$$Ax = b$$

where

$$A = \{a_{ij}\}_{i,j=1}^{n}, \quad x = \{x_i\}_{i=1}^{n}, \quad b = \{b_i\}_{i=1}^{n},$$

or

$$\begin{bmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{n1} & . & . & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}.$$

Your HW2 has a coding exercise that carries out this multiplication.

5.2 Direct Method: Gaussian Elimination

5.2.1 Simple system

Example 5.1. Consider the 2×2 system:

$$\begin{aligned} 3x_1 + 2x_2 &= 8 \\ 2x_1 - 3x_2 &= 1 \end{aligned}$$

or in matrix form

$$\begin{bmatrix} 3 & 2 \\ 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \end{bmatrix}.$$
$$Ax = b$$

(By inspection, the true solution is $x_1 = 2$ and $x_2 = 1$.)

1. Naive substitution ($O(n^4)$ method, avoid at all cost). See notes. Optional reading.

2. Gaussian elimination via **elementary row operations** and back substitution.

   Solution. First, we form the augmented matrix by appending b to the right hand side of A.

   $$[A \mid b] = \left[\begin{array}{cc|c} 3 & 2 & 8 \\ 2 & -3 & 1 \end{array}\right].$$

   We multiply the first row by $2/3$ so that the first entry of each row matches. This modifies the above into

   $$\left[\begin{array}{cc|c} 3 & 2 & 8 \\ 2 & -3 & 1 \end{array}\right] \to \left[\begin{array}{cc|c} 2 & \frac{4}{3} & \frac{16}{3} \\ 2 & -3 & 1 \end{array}\right].$$

   Now, we create a new second row by subtracting the first from the second,

   $$\left[\begin{array}{cc|c} 2 & \frac{4}{3} & \frac{16}{3} \\ 2 & -3 & 1 \end{array}\right] \to \left[\begin{array}{cc|c} \boxed{2} & \frac{4}{3} & \frac{16}{3} \\ 0 & \boxed{\frac{13}{3}} & \frac{13}{3} \end{array}\right].$$

   This transformed matrix is now in **row echelon form**, meaning that the leading entry (the first nonzero entry) is to the right of the leading entry of the previous rows (see the boxed entries).

   From the last row, we can immediately read that

   $$x_2 = \frac{13/3}{13/3} = 1.$$

   Then, back substitution into the first equation gives

   $$x_1 = \frac{16/3 - \frac{4}{3}x_2}{2} = \frac{4}{2} = 2.$$

Remark 5.1. *The most important takeaway here is that the procedures in the naive substitution CANNOT be represented by* ***elementary row operations*** *and hence by updating the matrix representation of the system. The second method, however, always admits a matrix representation.*

5.2.2 Larger System

Example 5.2. (4×4 system)

$$\begin{aligned}
E_1: &\quad x_1 + x_2 \quad + 3x_4 = 4,\\
E_2: &\quad 2x_1 + x_2 - x_3 + x_4 = 1,\\
E_3: &\quad 3x_1 - x_2 - x_3 + 2x_4 = -3,\\
E_4: &\quad -x_1 + 2x_2 + 3x_3 - x_4 = 4.
\end{aligned}$$

We keep the first equation intact. Perform

$$\begin{aligned}
(E_2 - 2E_1) &\to (E_2)\\
(E_3 - 3E_1) &\to (E_3)\\
(E_4 + E_1) &\to (E_4)
\end{aligned}$$

to obtain

$$\begin{aligned}
E_1: &\quad x_1 + x_2 \quad + \ 3x_4 = \ \ 4,\\
E_2: &\quad -x_2 - \ x_3 + \ x_4 = -7,\\
E_3: &\quad -4x_2 - \ x_3 - 7x_4 = -15,\\
E_4: &\quad 3x_2 + 3x_3 + 2x_4 = \ \ 8.
\end{aligned}$$

Now, we start with the second row, and try to eliminate the coeffcients of x_2 in row 3 and row 4. Perform

$$\begin{aligned}
(E_3 - 4E_2) &\to (E_3)\\
(E_4 + 3E_2) &\to (E_4)
\end{aligned}$$

to obatin

$$\begin{aligned}
E_1: &\quad x_1 + x_2 \quad + \ \ 3x_4 = \ \ 4,\\
E_2: &\quad -x_2 - \ x_3 + \ \ x_4 = -7,\\
E_3: &\quad 3x_3 + 13x_4 = \ \ 13,\\
E_4: &\quad -13x_4 = \ \ 13.
\end{aligned}$$

In matrix form, this looks like

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ 13 \\ -13 \end{bmatrix}$$

where we observe the matrix is **upper triangular**. The unknowns now can be solved backwards starting with x_4. Indeed,

$$x_4 = \frac{-13}{-13} = 1,$$

$$x_3 = \frac{13 - 13x_4}{3} = 0,$$

$$x_2 = \frac{-7 - (-5)\,x_4 - (-1)\,x_3}{-1} = 2,$$

$$x_1 = \frac{4 - 3x_4 - 0x_3 - x_2}{1} = -1.$$

5.2.3 Errors Still Occur

Example 5.3. Consider the 3×3 system

$$\begin{aligned} E_1: \quad & -x_1 + 4x_2 + x_3 = 8 \\ E_2: \quad & \frac{5}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 = 1 \\ E_3: \quad & 2x_1 + x_2 + 4x_3 = 11 \end{aligned}$$

or

$$\begin{bmatrix} -1 & 4 & 1 \\ \frac{5}{3} & \frac{2}{3} & \frac{2}{3} \\ 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \\ 11 \end{bmatrix}$$

or in augmented form

$$\left[\begin{array}{ccc|c} -1 & 4 & 1 & 8 \\ \frac{5}{3} & \frac{2}{3} & \frac{2}{3} & 1 \\ 2 & 1 & 4 & 11 \end{array}\right].$$

The true solution is

$$x_1 = -1, \quad x_2 = 1, \quad x_3 = 3.$$

Let's solve this system using 2-digit rounding arithmetic and Gaussian elimination with backward substitution. We begin with elementary row operations on E_2 and E_3 using E_1.

- $\left(E_2 + \frac{5}{3}E_1\right) \to (E_2)$

$\frac{5}{3}E_1$ has exact entries,

$$\begin{array}{ccc|c} -\frac{5}{3} & \frac{20}{3} & \frac{5}{3} & \frac{40}{3} \end{array}$$

We need to 2-digit round here into

$$\begin{array}{ccc|c} -1.7 & 6.7 & 1.7 & 13 \end{array}$$

We also round the second row as

$$\begin{array}{cccc} -1.7 & 6.7 & 1.7 & 13 \\ +)1.7 & 0.67 & 0.67 & 1 \\ \hline 0 & 7.37 \to 7.4 & 2.37 \to 2.4 & 14 \end{array}$$

- $(E_3 + 2E_1) \to (E_3)$

We are lucky that E_3 does not require rounding initially. Nor does $2E_1$

$$\begin{array}{ccc|c} -2 & 8 & 2 & 16 \end{array}$$

$$\begin{array}{ccc|c} -2 & 8 & 2 & 16 \\ +)2 & 1 & 4 & 11 \\ \hline 0 & 9 & 6 & 27 \end{array}$$

After these two elementary row operations, we have

$$\left[\begin{array}{ccc|c} -1 & 4 & 1 & 8 \\ 0 & 7.4 & 2.4 & 14 \\ 0 & 9 & 6 & 27 \end{array}\right].$$

Now, we need to use E_2 to modify E_3 by killing the 9. We do

$$\left(E_3 - \frac{9}{7.4}E_2\right) \to (E_3)$$

Note that, with rounding, $9/7.4 = 1.2$, but this will NOT be applied to the first nonzero term because $7.4 \times \frac{9}{7.4} = 9$.

$$\frac{9}{7.4}E_2 = \left[\begin{array}{cccc} 0 & \frac{9}{7.4} \times 9 & 2.4 \times \cancel{\frac{9}{7.4}}^{1.2} & 14 \times \cancel{\frac{9}{7.4}}^{1.2} \end{array}\right] = \left[\begin{array}{cccc} 0 & 9 & 2.88 \to 2.9 & 16.8 \to 17 \end{array}\right]$$

Then,

$$E_3 - \frac{9}{7.4}E_2 = \begin{array}{ccc|c} 0 & 9 & 6 & 27 \\ -)0 & 9 & 2.9 & 17 \\ \hline 0 & 0 & 3.1 & 10 \end{array}$$

and we arrive at row echelon form

$$\left[\begin{array}{ccc|c} -1 & 4 & 1 & 8 \\ 0 & 7.4 & 2.4 & 14 \\ 0 & 0 & 3.1 & 10 \end{array}\right].$$

With back substitution, we have

$$x_3 = \frac{10}{3.1} = 3.2258 \to 3.2.$$

$$x_2 = \frac{14 - 2.4 \times 3.2}{7.4} = \frac{14 - 7.68 \to 7.7}{7.4} = \frac{6.3}{7.4} = 0.85135 \to 0.85.$$

$$x_1 = \frac{8 - x_3 - 4x_2}{-1} = \frac{8 - 3.2 - 4 \times 0.85}{-1} = \frac{8 - 3.2 - 3.4}{-1} = -1.2.$$

Compare to the true solution

	2-digit rounding	True
x_1	-1.2	-1
x_2	0.85	1
x_3	3.2	3

6 Lecture VI: Algorithm of Gaussian Elimination and Partial Pivoting

In the last lecture, we studied specific examples of linear systems and recognized the procedures of row-reduction, i.e., the process of Gaussian Elimination. The first step of the algorithm involves multiplying a specific row by a multiplier, which depends on the leading coefficient of other rows. One reduces the matrix until the last row has a nonzero leading coefficient at the last entry.

In this lecture, we generalize the procedure to an arbitrary $n \times n$ system. Stipulations arise, however, for general matrices, such as a particular row to be reduced does NOT have a leading nonzero entry on the diagonal. In this case, one must perform a row swap. We elucidate all such restrictions and their remedies in the following discussion, follwed by a thorough description of the algorithm.

6.1 Generalizations to $n \times n$ System

Recall from linear algebra that a system of equations with upper/lower triangular matrix representation is very easy to solve, just like the above. In general, consider the following triagnular system,

$$\begin{bmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ 0 & a_{22} & . & . & a_{2n} \\ . & 0 & * & * & * \\ . & . & 0 & * & * \\ 0 & . & . & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}. \tag{6.1}$$

where $*$ is a nonzero entry. We assume further that the diagonal entries are nonzero (to be relaxed later).

We can read off the $x_n = \frac{b_n}{a_{nn}}$ immediately. Then,

$$x_{n-1} = \frac{b_{n-1} - a_{(n-1)n}x_n}{a_{(n-1)(n-1)}}$$

can be found since we know x_n already. Pushing this sequence backwards, we can find all x_i's sequentially, that is,

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}$$

as we know about $x_{i+1}, \ldots x_n$ from previous steps. This procedure is called **back substitution**.

Therefore, it is very beneficial to reduce any augmented matrix $\tilde{A} = [A \mid b]$ into a upper triangular form considered above (Equation (6.1)). In other words, we reduce the augmented system to **row echelon form**, that is, for the k^{th} row, the first nonzero entry is to the right of the k^{th} column. More precisely, we achieve this by the following **elementary row operations.**

Consider the system

$$\begin{bmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & a_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{n1} & . & . & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}$$

which can be further represented by the **augmented** matrix,

$$\widetilde{A} = [A \mid b] = \left[\begin{array}{ccccc|c} a_{11} & a_{12} & . & . & a_{1n} & a_{1,n+1} \\ a_{21} & a_{22} & . & . & a_{2n} & a_{2,n+1} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{n1} & . & . & . & a_{nn} & a_{n,n+1} \end{array}\right]$$

where

$$a_{i,n+1} = b_i, \quad i = 1, 2, \ldots, n.$$

1. Given that $a_{11} \neq 0$, construct the multipliers

   $$m_{j1} = \frac{a_{j1}}{a_{11}}, \quad j = 2, 3, \ldots, n$$

   for rows other than row 1. We call a_{11} the **pivot** of the first row.

2. Eliminate the coefficient of x_1 in each row via:

   $$\left(E_j - \frac{a_{j1}}{a_{11}} E_1\right) \rightarrow (E_j), \quad j = 2, 3, \ldots, n, \tag{6.2}$$

   This will make the system look like

   $$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & . & . & a_{1n} & a_{1,n+1} \\ 0 & a_{22} & . & . & a_{2n} & a_{2,n+1} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & a_{n-1}a_2 & . & . & a_{nn} & a_{n,n+1} \end{array}\right]$$

   where the entries in the second row and below are NOT necessarily the same as before. This is simply showing the resulting **structure** after applying Equation (6.2).

3. For each remaining coefficients, we perform, by keeping E_i intact but using it to modify other rows, i.e.

$$\left(E_j - \frac{a_{ji}}{a_{ii}} E_i\right) \to (E_j), \quad j = i+1, i+2, \ldots, n,$$

provided that $a_{ii} \neq 0$. This eliminates the coefficient of x_i in each row below the i^{th} for $i = 1, 2, \ldots, n-1$.

6.2 Algorithm

1. For $i = 1, \ldots, n-1$

   (a) Let p be the smallest integer with $i \leqslant p \leqslant n$ and $a_{pi} \neq 0$
   If no integer p can be found
   then OUTPUT ('no unique solution exists');
   STOP.
   (This step is checking if we find an entire row of zeros).

   (b) If $p \neq i$, then perform $(E_p) \leftrightarrow (E_i)$.
   (Say, for $i = 1$, we found $p = 2$, meaning that $a_{11} = 0$ while $a_{12} \neq 0$. This row thus cannot be used to perform row operations since the multiplier requires that $a_{11} \neq 0$. However, since there is only one zero to the left of the leading entry a_{12}, it would be nice if it is moved to the second row, such that it is "nicely reduced" already.)

   (c) For $j = i+1, \ldots, n$

      i. Set $m_{ji} = \frac{a_{ji}}{a_{ii}}$ (form multiplier used on E_i to modify E_j via **elementary row operations**);

      ii. Perform $(E_j - m_{ji}E_i) \to (E_i)$ (**elementary row operations**).

2. If $a_{nn} = 0$, then OUTPUT ('no unique solution exists');
   STOP.

3. Set $x_n = \frac{a_{n,n+1}}{a_{nn}}$. (Start back substitution)

4. For $i = n-1, \ldots, 1$, set

$$x_i = \frac{a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}.$$

5. OUTPUT $(x_1, \ldots, x_n)$.

6.3 Operation Counts

Suppose, we are trying to use the i^{th} row to remove the coefficient of x_i in all subsequent rows ($i+1, i+2$ up to n), what does the "modified" matrix look like before we begin?

$$\widetilde{A}^{(i)} = \left[\begin{array}{ccccccccccc|c} a_{11} & a_{12} & a_{13} & . & . & a_{1,i-1} & a_{1i} & . & . & . & . & a_{1n} & a_{1,n+1} \\ 0 & a_{22} & a_{23} & . & . & a_{2,i-1} & a_{2i} & . & . & . & . & a_{2n} & a_{2,n+1} \\ 0 & 0 & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & a_{i-1,i-1} & . & . & . & . & . & . & . \\ . & . & . & . & . & 0 & \boxed{a_{ii}} & a_{i,i+1} & . & . & . & a_{in} & a_{i,n+1} \\ . & . & . & . & . & 0 & a_{i+1,i} & a_{i+1,i+1} & . & . & . & a_{i+1,n} & a_{i+1,n+1} \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & 0 & a_{ni} & a_{n,i+1} & . & . & . & a_{nn} & a_{n,n+1} \end{array}\right].$$

So, in the i^{th} row, we have $(n+1)-(i-1) = n-i+2$ entries (assuming $a_{ii} \neq 0$); a sanity check for whether this formula is correct is by checking $i = 1$ – we must have $n-1+2 = n+1$ entries to begin with.

For each row E_j below, we need to find the multiplier,

$$m_{ji} = \frac{a_{ji}}{a_{ii}}, \quad j = i+1, i+2, \ldots, n,$$

that is, a total of $(n-i)$ divisions. Carrying out the multiplications for each j, i.e.

$$m_{ji} E_i,$$

we performed $(n-i+1)$ multiplications, where we do not count the first entry – it is always going to be a_{ji}. To perform this multiplication for each j, we have done

$$(n-i)(n-i+1)$$

multiplications. Therefore, to finish Gaussian elimination pivoted at a_{ii}, we perform a total of multiplication/divisions

$$(n-i) + (n-i)(n-i+1) = (n-i)(n-i+2)$$

times.

Then, $E_j - m_{ji}E_i$ does $(n-i+1)$ subtractions in each row, and does this $(n-i)$ times, that is,

$$(n-i+1)(n-i)$$

additions/subtractions.

Thus, to find the total number of multiplications and divisions, we add for $i = 1, \ldots, n-1$, in the sum

$$\begin{aligned}
\sum_{i=1}^{n-1}(n-i)(n-i+2) &= \sum_{i=1}^{n-1}(n-i)^2 + 2\sum_{i=1}^{n-1}(n-i) \\
&= \sum_{i=1}^{n-1} i^2 + 2\sum_{i=1}^{n-1} i \\
&= \frac{(n-1)\,n\,(2n-1)}{6} + 2\frac{(n-1)\,n}{2} \\
&= \frac{2n^3+3n^2-5n}{6}.
\end{aligned}$$

Similarly, total number of additions/subtractions is

$$\begin{aligned}
\sum_{i=1}^{n-1}(n-i)(n-i+1) &= \sum_{i=1}^{n-1}(n-i)^2 + \sum_{i=1}^{n-1}(n-i) \\
&= \sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i \\
&= \frac{(n-1)\,n\,(2n-1)}{6} + \frac{(n-1)\,n}{2} \\
&= \frac{n^3-n}{3}.
\end{aligned}$$

In backward substitution, the n^{th} requires one mere division, $x_{nn} = \frac{a_{n,n+1}}{a_{nn}}$. For the i^{th} row, we perform

$$x_i = \frac{a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}$$

which incurs $(n-(i+1)+1) = n-i$ multiplications to carry out $a_{ij}x_j$, $(n-(i+1))$ additions in the sum, one subtraction and then one division. We perform i from $n-1$ down to 1.

More precisely, in the backward substitution step, the total number of multiplica-

tion is

$$\begin{aligned}
1+\sum_{i=1}^{n-1}(n-i+1) &= 1+\sum_{i=1}^{n-1}(n-i)+(n-1) \\
&= n+\sum_{i=1}^{n-1} i \\
&= n+\frac{n(n-1)}{2} \\
&= \frac{n^2+n}{2}.
\end{aligned}$$

The total number of addition and subtractions is

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

Altogether, with elimination AND substitution, we have

$$\begin{aligned}
\text{(Multiplications/divisions)} \quad & \frac{2n^3+3n^2-5n}{6}+\frac{n^2+n}{2} = \frac{n^3}{3}+n^2-\frac{n}{3}, \\
\text{(Additions/subtractions)} \quad & \frac{n^3-n}{3}+\frac{n^2-n}{2} = \frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}.
\end{aligned}$$

We can see that for n large, total number of flops is $O(n^3)$.

6.4 Gaussian Elimination with Partial Pivoting

Numerical instability occurs when we divide by a small number. The large quotient, if used, produces large round-off errors. Consider the multiplier step in Gaussian elimination,

$$m_{ji} = \frac{a_{ji}}{a_{ii}}.$$

What if $a_{ii} \ll 1$, and as result, m_{ji} is very large. Then, m_{ji} is multiplied to each term of E_i which may result in large round-off error (remember, eps (single (2^{40}))'s neighbor is some 10^5 units away). What's more, in back substitution, we have

$$x_i = \frac{a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}}{a_{ii}}$$

where a very small a_{ii} can lead to a large round-off error.

However, dividing by a **big** number is completely O.K. because the resulting small number is very fine machine representation, i.e., small round-off error. This desire to achieve numerical stability prompts the following modified version of Gaussian elimination.

Let's begin with a motivating example.

Example 6.1. We first look at an example in which round-off errors occur when the leading entry of a system is too small.

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17; \\ 5.291x_1 - 6.130x_2 &= 46.78. \end{aligned}$$

Though the numbers look quite contrived, we can spot the exact solution as

$$\begin{aligned} x_1 &= 10.00; \\ x_2 &= 1.000. \end{aligned}$$

Now, let's proceed with 4-digit rounding arithmetic. The multiplier for the second row is

$$m_{21} = \frac{a_{21}}{a_{11}} = \frac{5.291}{0.003000} = 1763.6\overline{6} = 1764$$

after rounding. Performing $(E_2 - m_{21}E_1) \to (E_2)$ and the appropriate rounding yields

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &\approx 59.17; \\ -104300x_2 &\approx -104400, \end{aligned}$$

instead of the exact system

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17; \\ -104309.37\overline{6}x_2 &= 104309.37\overline{6}. \end{aligned}$$

The round-off error has already infected $m_{21}a_{13}$ and a_{23}. It is now propagating through the back substitution step

$$x_2 \approx 1.001$$

which is an OK approximation to the true solution $x_2 = 1$. The real trouble arises when back substituting for x_1, i.e.,

$$x_1 \approx \frac{59.17 - (59.14)\,(1.001)}{0.003000} = -10.00.$$

Woah, this is opposite of what we knew for x_1.

So, what's a good remedy? Swapping the rows, we get

$$\begin{aligned} 5.291x_1 - 6.130x_2 &= 46.78; \\ 0.003000x_1 + 59.14x_2 &= 59.17. \end{aligned}$$

Now,

$$m_{21} = \frac{a_{21}}{a_{11}} = \frac{0.003000}{5.291} = 0.0005670$$

after rounding to 4-digit. Then, $(E_2 - m_{21}E_1) \to (E_2)$ gives

$$\begin{aligned} 5.291x_1 - 6.130x_2 &= 46.78; \\ 59.14x_2 &= 59.14. \end{aligned}$$

We retrieve $x_2 = 1.000$ and $x_1 = 10.00$ exactly, avoiding the catastrophe seen earlier.

So, that was a 2×2 system. How about larger ones? What is the general procedure to avoid dividing by small numbers and producing multipliers that lead to large round-off errors?

6.5 Gaussian Elimination with Partial Pivoting: Algorithm

Suppose we are now using the k^{th} row to perform Gaussian elimination for the rows below. If a_{kk} is small relative to the entries a_{ij} for $k \leqslant i \leqslant n$ and $k \leqslant j \leqslant n$, pivoting is performed by selecting an element a_{pq} with a larger magnitude as the pivot and interchanging the k^{th} and p^{th} rows (and swapping the k^{th} and q^{th} columns is also viable, if necessary).

More precisely, this strategy, called **partial pivoting**, is to select an element in the same column (the k^{th} column) that is below the diagonal and has the largest absolute value; we determine the smallest $p \geqslant k$ such that

$$|a_{pk}| = \max_{k \leqslant i \leqslant n} |a_{ik}|$$

(remember, k is fixed here since we are using the k^{th} row to row reduce; the maximum we are searching is over the row index, which means, throw all entries in the k^{th} column below the diagonal).

1. For $i = 1, \ldots, n$, set $NROW(i) = i$. (Initialize row pointer – keeping the book for swapped rows).

2. For $i = 1, \ldots, n-1$ (elimination)

   (a) Let p be the smallest integer with $i \leqslant p \leqslant n$ and

   $$|a(NROW(p), i)| = \max_{i \leqslant j \leqslant n} |a(NROW(j), i)|$$

   where

   $$a(NROW(j), i) = a_{NROW(j),i}.$$

   This step is checking through the i^{th} column and looking for which entry we should use as the pivot. This step automatically completes the "zero checking" step in the algorithm for Naive Gaussian Elimination.

   (b) If $a(NROW(p), i) = 0$, then OUTPUT('no unique solution exists'); STOP. This step simply find all zeros underneath the diagonal, indicating more than one variable is free and we expect no unique solutions, e.g., for $i = 2$,

   $$\left[\begin{array}{ccc|c} 1 & 3 & 1 & 1 \\ 0 & 0 & 5 & 2 \\ 0 & 0 & 9 & 3 \end{array}\right]$$

   where we now are focusing on the second row and finding $a(NROW(p), 2) = 0$.

   (c) If $NROW(i) \neq NROW(p)$ (if the current leading entry is not the largest), then

   $$\begin{aligned} NCOPY &= NROW(i); \\ NROW(i) &= NROW(p); \\ NROW(p) &= NCOPY. \end{aligned}$$

   This step updates the list of swapped rows. You may think about why $NCOPY$ is introduced here.

(d) For $j = i+1, \ldots, n$,

i. Set

$$m\left(NROW\left(j\right), i\right) = \frac{a\left(NROW\left(j\right), i\right)}{a\left(NROW\left(i\right), i\right)}.$$

ii. Perform

$$\left(E_{NROW(j)} - m\left(NROW\left(j\right), i\right) \cdot E_{NROW(i)}\right) \rightarrow \left(E_{m(NROW(j))}\right)$$

3. If $a\left(NROW\left(n\right), n\right) = 0$, then OUTPUT('no unique solution exists'); STOP.

4. Set

$$x_n = \frac{a\left(NROW\left(n\right), n+1\right)}{a\left(NROW\left(n\right), n\right)}$$

(back substitution).

5. For $i = n-1, \ldots, 1$, set

$$x_i = \frac{a\left(NROW\left(i\right), n+1\right) - \sum_{j=i+1}^{n} a\left(NROW\left(i\right), j\right) x_j}{a\left(NROW\left(i\right), i\right)}.$$

6. OUTPUT $\left(x_1, \ldots, x_n\right)$.STOP.

Remark 6.1. *You may check the row reduced matrix after completing step* 3. *If you print out the matrix, you may notice that it is not exactly upper triangular. However, if you do print out the matrix with row numbers following the swapped indices, it must be upper triangular.*

Remark 6.2. *It is costly to* ***actually*** *swap rows. Instead, it is much less expensive to keep a list of swapped indices, and call upon them when needed. As long as this list is updated, we can proceed without affecting the solutions.*

Takeaway 6.1. (Recap)
The main point of partial pivoting is to reduce round-off errors, caused by multiplying possibly large multipliers (obtained from dividing small pivots). We avoid these errors by choosing the pivot carefully for each row.

6.6 Optional reading: Complete Pivoting

Complete pivoting involves checking for the maximal element in magnitude in both the k^{th} row **and** k^{th} column. Therefore, the procedures will be a mix of column and row swaps. While row swaps do not affect the order of the solution vector $(x_1, \ldots, x_n)$,

column swaps do, namely, for a swapped p^{th} and q^{th} column, we must swap the indices in the solution

$$(x_1, \ldots, x_p, \ldots, x_q, \ldots x_n) \rightarrow (x_1, \ldots, x_q, \ldots, x_p, \ldots x_n)\,.$$

Performing complete pivoting thus involves keeping a list of swapped columns that will eventually be used to reorder the output at the end of back substitution.

Noting all the additional procedures needed for complete pivoting, we recommend this method only for systems that demand so high of an accuracy that this increase in execution time can be justified.

7 Lecture VII: LU Decomposition

Gaussian Elimination with or without partial pivoting is a great tool to solve linear systems. The row swaps can sometimes be one stone for two birds (choosing a large **and** nonzero pivot). However, in many applications, such as high dimensional optimization problems, or partial differential equations that model physical systems, a system of $Ax = b$ needs to be solved many times, with a fixed A, but various b's. While Gaussian Elimination still applies, it does not scale well with a large amount of b's ($O(rn^3)$ if A is n-by-n and there are r of those b's). At the same time, we are always using the **same** A over and over again. Is there a better way of representing A so that we don't need to perform the elimination step repeatedly?

7.1 A Motivating Example

Consider the 2-by-2 system

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

The Gaussian Elimination step is

$$m_{21} = -\frac{a_{21}}{a_{11}} = -3$$

and thus

$$\left[\begin{array}{cc|c} 1 & 2 & 5 \\ 3 & 4 & 6 \end{array}\right] \overset{E_2 - 3E_1 \to E_2}{\Longrightarrow} \left[\begin{array}{cc|c} 1 & 2 & 5 \\ 0 & -2 & -9 \end{array}\right].$$

Our goal now is to recognize the very action of the multiplier determination m_{21} and the multiplication step $3E_1$. Notice that the operation is linear, in the sense that every entry in the second row is **multiplier** by a number, while the first row is **untouched**, and then two rows are added.

Let's focus on the "**untouched**" part. Is there a matrix M such that $MA = A$, namely, A is unchanged after multiplying M on the left? Yes, it is called the identity matrix, i.e., $M = I$.

$$I\,[A \mid b] := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \left[\begin{array}{cc|c} 1 & 2 & 5 \\ 3 & 4 & 6 \end{array}\right] = \left[\begin{array}{cc|c} 1 & 2 & 5 \\ 3 & 4 & 6 \end{array}\right]$$

Now, what about the **multiplier** part? Is there a matrix M such that MA multiplies the second row by -3 and zeros everything out?

$$M\,[A \mid b] := \begin{bmatrix} 0 & 0 \\ -3 & 0 \end{bmatrix} \left[\begin{array}{cc|c} 1 & 2 & 5 \\ 3 & 4 & 6 \end{array}\right] = \left[\begin{array}{cc|c} 0 & 0 & 0 \\ -3 & -6 & -15 \end{array}\right]$$

Piecing both components together, we have

$$I\left[A \mid b\right] + M\left[A \mid b\right] = (I + M)\left[A \mid b\right] = \left[\begin{array}{cc|c} 1 & 2 & 5 \\ 0 & -2 & -9 \end{array}\right]$$

where we define the **Lower Triangular Matrix** $\widetilde{L}$ as

$$\widetilde{L} := I + M = \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix}$$

Notice that the multiplier we found is conveniently stored in this matrix, in the off-diagonal entries. We further define the **Upper Triangular Matrix** U as

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

Ultimately, this observation led us to write

$$\widetilde{L}A = U \implies A = LU$$

where $L = \widetilde{L}^{-1}$ is also a Lower Triangular Matrix (proof required). We find that the inverse of a lower triangular matrix satisfies

$$\widetilde{L} = \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \implies L = \widetilde{L}^{-1} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}$$

To verify that $LU = A$, we compute explicitly that

$$LU = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = A$$

7.2 General LU Decomposition

In general, for an n-by-n matrix, the ith step of Gaussian Elimination produces a lower triangular matrix L_i, which suggests that

$$\widetilde{L}_{n-1}\widetilde{L}_{n-2}\ldots\widetilde{L}_2\widetilde{L}_1 A = U$$

Since the product of lower triangular matrices (and its inverse) is still lower triangular, the equation above suggests that there exists some lower triangular L and upper triangular U that decomposes A

$$LU = A.$$

In fact, we can identify precisely what each L_i is.

$$\widetilde{L}^{(i)} = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & . & . & . & 0 \\ 0 & 1 & . & . & . & . & . & . & . & . & . & . \\ . & 0 & 1 & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & 0 & 1 & . & . & . & . & . & . \\ . & . & . & . & . & m_{i+1,i} & 1 & . & . & . & . & . \\ . & . & . & . & . & m_{i+2,i} & . & 1 & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & . & . & 0 & m_{ni} & 0 & 0 & . & . & 0 & 1 \end{bmatrix}.$$

that is, the identity plus a matrix with only the ith column with the multipliers as entries below the diagonal.

7.3 Implementation and Complexity Advantage

In the literature, the LU decomposition is sometimes called **triangular decomposition**, for an apt and obvious reason. The use of this decomposition is immense when $Ax = b$ needs to be solved repeatedly for various b's. More precisely, when one determines the component L and U (via Gaussian Elimination, which is $O(n^3)$), one now performs the following substitution steps

$$Ax = b \implies LUx = b.$$

1. Let $z = Ux$, and solve $Lz = b$ via **forward substitution** ($O(n^2)$).
2. Then solve $Ux = z$ via **backward substitution** ($O(n^2)$).

So, if one is tasked to solve $Ax = b$ repeatedly for r different b's, one only needs to express the decomposition one time, instead of doing Gaussian elimination r times. Altogether, the operation count is

$$\underbrace{O\left(n^3\right)}_{\text{one step of GE to find factorization}} + \underbrace{O\left(rn^2\right)}_{r \text{ steps of substitution}}$$

rather than $O(rn^3)$ incurred by doing GE r times.

7.4 Examples

Consider the 3-by-3 system

$$Ax = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 9 \end{bmatrix} = b$$

Determine the LU decomposition of A and then use it to solve for x (though we can inspect a solution $(1, -1, 1)$).

We perform the Naive Gaussian Elimination (without pivoting).

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 9 \end{bmatrix}$$

$$L_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -7 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{bmatrix}$$

where

$$L_1 A = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix}$$

Now, by looking at the multiplier again, we see that

$$L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix}$$

and thus

$$L_2 L_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = U$$

Thus,

$$A = LU$$

where

$$L = (L_2 L_1)^{-1} = L_1^{-1} L_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix}$$

where we noted that the inverse of a lower triangular matrix is simply the lower part with flipped signs.

We solve the forward system $Lz = b$ using forward substitution,

$$Lz = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 9 \end{bmatrix} = b$$
$$\implies z_1 = 2, \quad z_2 = 5 - 4z_1 = -3, \quad z_3 = 9 - 7z_1 - 2z_2 = -1$$

Then, we solve $Ux = z$ using backward substitution,

$$Ux = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix} = z$$
$$\implies x_3 = 1, \quad x_2 = \frac{-3 - (-6)\, x_3}{-3} = -1, \quad x_1 = \frac{2 - 2x_2 - 3x_3}{1} = 1$$

as inspected.

7.5 Warning about the Inverse of Lower Triangular Matrices

We are blessed with having 1's on the diagonal of L_i's so that the inverse is easier to read off. For **general** lower triangular matrices, this "reading off" is in general not true, and in fact, should be taken with extreme care. The following theorem is a result about the inverse of the L_i's we used just now.

Theorem 7.1. *Given a lower triangular matrix of the form*

$$L = \begin{bmatrix} 1 & 0 & . & . & . & . & . & 0 \\ 0 & 1 & 0 & . & . & . & . & 0 \\ 0 & 0 & 1 & 0 & . & . & . & . \\ . & 0 & . & . & . & . & . & . \\ . & . & . & a_{i+1,i} & . & . & . & . \\ . & . & . & * & . & . & . & . \\ . & . & . & * & . & . & 1 & 0 \\ 0 & 0 & . & a_{ni} & . & . & 0 & 1 \end{bmatrix}$$

where i is an arbitrary column, with nonzero entries under the diagonal, then the

inverse is then given by

$$L^{-1} = \begin{bmatrix} 1 & 0 & . & . & . & . & . & 0 \\ 0 & 1 & 0 & . & . & . & . & 0 \\ 0 & 0 & 1 & 0 & . & . & . & . \\ . & 0 & . & . & . & . & . & . \\ . & . & . & -a_{i+1,i} & . & . & . & . \\ . & . & . & * & . & . & . & . \\ . & . & . & * & . & . & 1 & 0 \\ 0 & 0 & . & -a_{ni} & . & . & 0 & 1 \end{bmatrix}$$

Proof. Note that $L = I + M$ where M is the matrix with just the nonzero subcolumn in an arbitrary column below the diagonal. The inverse then is $L^{-1} = (I + M)^{-1}$. At the same time, we note that $M^2 = 0$ (nilpotent matrix of order 2, just verify it yourself by explicit computation). Thus,

$$(I + M)(I - M) = I + M - M - M^2 = I$$

showing that $(I + M)$ and $(I - M)$ are inverse pairs of each other. Thus,

$$L^{-1} = I - M,$$

i.e., preserve the diagonal with 1's, and set the off-diagonal terms negative. □

There is a more general theorem about nilpotent matrices of arbitrary order k, namely, $M^k = 0$. Then the inverse matrix of $(I + M)$ has a rather elegant geometric series representation. You can read it here.

7.6 PLU Decomposition: Accounting for Partial Pivoting

Given a matrix A, is there a matrix PA such that the resulting product has the ith and jth row swapped?

Example 7.1. Consider the square matrix

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}.$$

We search through the first column and locate the row in which the largest (in magnitude) column entry resides and perform partial pivoting. Here, we identify that

the third row and the first should swap,

$$\widehat{A} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

where now, dividing by 8 to find the multiplier becomes much more stable.

This procedure can be represented by the permutation operation on A, as in

$$P^{(1)}A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

where we note that the second and the fourth row stay put since the second and fourth diagonal entry remain "on". The 1st row goes to the 3rd after being multiplied by $(0, 0, 1, 0)$, and similarly for the 3rd row to go to the 1st.

In Gaussian Elimination with partial pivoting, row swaps almost always occur since we search for the largest element (in magnitude) in a column below the diagonal. A permutation matrix P is multiplied on the left of A (or any intermediate steps) to reflect the swap. Since we do row swaps nearly at every step, the actions on A can be written as

$$L_n P_n L_{n-1} P_{n-1} \dots L_2 P_2 L_1 P_1 A = U$$

This does NOT seem so friendly as opposed to the naive LU decomposition. However, because of the fact that permutation matrices are their own inverses (swapping twice undoes the swap) and another brilliant observation (see remark below),, we are able to rewrite

$$L'_n L'_{n-1} \dots L'_1 P_n P_{n-1} \dots P_1 = L_n P_n L_{n-1} P_{n-1} \dots L_2 P_2 L_1 P_1$$

which then implies that

$$PA = LU$$

where $P = P_{n-1} P_n \cdots P_2 P_1$ is the (aggregate) permutation matrix. This decomposition is known as the **PLU Decomposition**.

Remark 7.1. *Did we just get lucky that the product of permutation matrices gives us what we wanted? This fact is not obvious, but the details may be too complicated for the course. A great explanation is provided in Trefethen and Bau (page 159 - 160).*

8 Lecture VIII: Cholesky Decomposition

So far, we have developed Gaussian elimination and its matrix formulation, the LU decomposition, as general-purpose tools for solving linear systems $Ax = b$. A key advantage of these methods is that once the matrix A has been factorized, the same decomposition can be reused to efficiently solve multiple systems with different right-hand sides.

These methods apply to arbitrary matrices, but they make no attempt to exploit additional structure in A. In practice, many important linear systems involve matrices that are *symmetric*. Gaussian elimination will still work in this case, but it treats symmetry as accidental rather than informative, and therefore misses opportunities for further simplification and efficiency. This naturally leads to the following question:

> *If A is symmetric, can we solve $Ax = b$ more efficiently by exploiting this structure?*

8.1 A motivating question: when does $A = LL^{\mathrm{T}}$ make sense?

A particularly appealing idea is to look for a factorization of the form

$$A = LL^{\mathrm{T}},$$

where L is lower triangular. Such a factorization would automatically respect symmetry and would reduce both storage and computational cost compared to LU decomposition.

However, it is not immediately clear:

- whether such a factorization always exists, and
- what properties of A would make it possible.

To investigate this, we begin with a concrete example.

8.2 Motivating example

Consider a 3×3 symmetric matrix

$$A = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

We ask whether it is possible to find a lower triangular matrix

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

such that $A = LL^{\mathrm{T}}$.

Carrying out the matrix multiplication gives

$$A = LL^{\mathrm{T}} = \begin{bmatrix} l_{11}^2 & * & * \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & * \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{bmatrix},$$

where the asterisks indicate symmetric entries.

Matching coefficients leads to the system

$$\begin{aligned} l_{11} &= \sqrt{a_{11}}, \\ l_{21} &= \frac{a_{21}}{l_{11}}, \\ l_{22} &= \sqrt{a_{22} - l_{21}^2}, \\ l_{31} &= \frac{a_{31}}{l_{11}}, \\ l_{32} &= \frac{a_{32} - l_{31}l_{21}}{l_{22}}, \\ l_{33} &= \sqrt{a_{33} - l_{31}^2 - l_{32}^2}. \end{aligned}$$

At first glance, this looks promising: the entries of L can be computed sequentially, much like LU decomposition.

However, a closer look reveals potential problems:

- divisions by l_{11} and l_{22} require these quantities to be nonzero,
- square roots require the expressions underneath to be nonnegative.

This leads to a more fundamental question:

What property of A guarantees that all these quantities are well-defined?

8.3 Positive definiteness

The answer turns out to be a property that is both algebraic and geometric.

Definition 8.1 (Positive definite matrix)**.** *An $n \times n$ symmetric real matrix M is called* ***positive definite*** *if*

$$x^{\mathrm{T}} M x > 0 \quad \textit{for all nonzero } x \in \mathbb{R}^n.$$

This condition has an important interpretation: it means that the quadratic form $x^{\mathrm{T}} M x$ behaves like a squared length or energy. In particular, it can never be negative and vanishes only when $x = 0$.

It is not a coincidence that positive definiteness resolves the difficulties encountered above. If A is positive definite, then each diagonal quantity that appears in the construction of L is strictly positive, ensuring that all square roots and divisions are valid. Don't believe this? See the next optional section.

Conversely, if $A = LL^{\mathrm{T}}$ for some real lower triangular L with positive diagonal entries, then

$$x^{\mathrm{T}} A x = \|L^{\mathrm{T}} x\|_2^2 > 0 \quad \text{for } x \neq 0,$$

so A must be positive definite. Not sure what $\|\cdot\|$ means? See the section after next for a primer. We will learn more about this next week.

Conclusion/Theorem: A symmetric matrix admits a factorization $A = LL^{\mathrm{T}}$ if and only if it is positive definite. This factorization is called the **Cholesky decomposition**.

Optional: why are the diagonal quantities in Cholesky strictly positive? In the Cholesky formulas,

$$l_{11} = \sqrt{a_{11}}, \qquad l_{22} = \sqrt{a_{22} - l_{21}^2}, \qquad \ldots, \qquad l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2},$$

why are we allowed to take these square roots? In other words, why are the quantities under the radicals strictly positive when A is positive definite?

Step 1: the case $i = 1$. If A is positive definite, then $x^{\mathrm{T}} A x > 0$ for every nonzero x. Take $x = e_1 = (1, 0, \ldots, 0)^{\mathrm{T}}$. Then

$$e_1^{\mathrm{T}} A e_1 = a_{11} > 0.$$

Hence $l_{11} = \sqrt{a_{11}}$ is well-defined and nonzero.

Step 2: the case $i = 2$ (fully multiplied out). Now take a vector supported on the first two coordinates,

$$x = \begin{bmatrix} t \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad t \in \mathbb{R}.$$

Then the quadratic form becomes a quadratic polynomial in t:

$$x^{\mathrm{T}} A x = \begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = a_{11} t^2 + 2a_{21} t + a_{22}.$$

Positive definiteness of A implies $x^{\mathrm{T}} A x > 0$ for every $t \in \mathbb{R}$, so the quadratic

$$q(t) = a_{11} t^2 + 2a_{21} t + a_{22}$$

must be strictly positive for all real t.

Since we already know $a_{11} > 0$, the only way a parabola with leading coefficient $a_{11} > 0$ can be positive for all t is if it has *no real roots*, i.e. its discriminant is negative:

$$(2a_{21})^2 - 4a_{11}a_{22} < 0 \quad \Longleftrightarrow \quad a_{11}a_{22} - a_{21}^2 > 0.$$

Now compare this to the Cholesky computation for the 2×2 leading block:

$$l_{21} = \frac{a_{21}}{l_{11}} = \frac{a_{21}}{\sqrt{a_{11}}}, \qquad l_{22}^2 = a_{22} - l_{21}^2 = a_{22} - \frac{a_{21}^2}{a_{11}} = \frac{a_{11}a_{22} - a_{21}^2}{a_{11}}.$$

We have shown $a_{11}a_{22} - a_{21}^2 > 0$ and also $a_{11} > 0$, therefore

$$l_{22}^2 = \frac{a_{11}a_{22} - a_{21}^2}{a_{11}} > 0,$$

so $l_{22} = \sqrt{l_{22}^2}$ is well-defined and nonzero. This explains, in a concrete computation, why the diagonal step in Cholesky does not break down for a positive definite matrix.

Remark (what changes for general i). For larger i, one repeats the same idea: choose vectors supported on the first i coordinates, expand $x^{\mathrm{T}} A x$ as a quadratic form in a free parameter, and deduce that the relevant "residual" quantity (the expression under the square root) must be strictly positive. In more advanced language, positive definiteness implies each leading principal submatrix is positive definite, and the Cholesky diagonal terms are exactly the successive positive "Schur complements" that appear in this process.

A brief note on vector norms. For a vector $x \in \mathbb{R}^n$, the **Euclidean (or 2-)norm** is defined as

$$\|x\|_2 := \sqrt{x^{\mathrm{T}} x}.$$

Geometrically, $\|x\|_2$ represents the length of the vector x in $\mathbb{R}^n$. In particular,

$$\|x\|_2^2 = x^{\mathrm{T}} x.$$

We will study vector and matrix norms in detail later. For now, this definition allows us to interpret expressions such as $x^{\mathrm{T}} A x$ when $A = LL^{\mathrm{T}}$ as a squared length:

$$x^{\mathrm{T}} A x = x^{\mathrm{T}} L L^{\mathrm{T}} x = \|L^{\mathrm{T}} x\|_2^2.$$

8.4 The Cholesky Algorithm

We now describe how to compute the Cholesky factor L for a general $n \times n$ positive definite matrix A. In general, for an n-by-n positive definite matrix, we consider

$$A = LL^T = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ l_{n1} & \cdots & l_{n,n-1} & \cdot \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & & l_{n,n-1} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & l_{nn} \end{bmatrix}$$

We first easily identify that

$$l_{11} = \sqrt{a_{11}}.$$

On the diagonal of L, we identify the equation (gleaning from the 3-by-3 example)

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$$

Now, for an arbitrary entry a_{ij}, we utilize the symmetry and find

$$\begin{aligned} a_{ij} = \left(LL^T\right)_{ij} &\overset{\text{definition of matrix multiplication}}{=} \sum_{k=1}^{n} l_{ik} l_{kj}^T \\ &\overset{l_{ik}=0 \text{ for } k>i \text{ since } L \text{ is lower triangular}}{=} \sum_{k=1}^{i} l_{ik} l_{kj}^T \\ &\overset{\text{transpose}}{=} \sum_{k=1}^{i} l_{ik} l_{jk} \\ &\overset{\text{extract the diagonal term}}{=} l_{ii} l_{ji} + \sum_{k=1}^{i-1} l_{ik} l_{jk} \end{aligned}$$

This implies that

$$l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}}{l_{ii}}. \tag{8.1}$$

These equations won't be solvable until we recognize which ones need to be solved first. Note that for a fixed column i, Equation (8.1) needs the values from l_{jk} for $k = 1, \ldots, i-1$, which is only sequentially found from l_{j1} up to l_{ji} (i.e., columns prior to i); moreover, we need the diagonal entry l_{ii}, which should be found first when starting at the fixed column i. Therefore, in order to determine all entries in L, we must intertwine the diagonal and the off-diagonal element computation. The exact algorithm is given as follows:

INPUT: dimension n; entries a_{ij}.
OUTPUT: the entries l_{ij}.

1. Set $l_{11} = \sqrt{a_{11}}$.
2. For $j = 2, \ldots, n$, set $l_{j1} = a_{j1}/l_{11}$. (This step fills the first column)
3. For $i = 2, \ldots, n-1$,
   (a) Set $l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$. (Diagonal of the ith column)
   (b) For $j = i+1, \ldots, n$, (below the diagonal in th ith column) set
   $$l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}}{l_{ii}}.$$
4. Set $l_{nn} = \sqrt{a_{nn} - \sum_{k=1}^{n-1} l_{nk}^2}$

5. **OUTPUT** l_{ij}.

This algorithm, known as the **Cholesky-Crout algorithm**, is completed column-by-column. However, if we observe the 3-by-3 example carefully, one can also achieve the same result by going row-by-row (the **Cholesky–Banachiewicz algorithm**).

So, is there a real-world problem where Cholesky decomposition is useful? See in the next lecture!

9 Lecture IX: The Least Square Problem

In the last lecture, we constructed the Cholesky decomposition of a positive definite matrix, inspired by the fundamental LU decomposition. A natural question arises: is there a real-world problem where a system of equations is written in terms of a (symmetric) positive definite matrix?

9.1 Problem Setup

Consider observation data $b_1, b_2, \ldots, b_m \in \mathbb{R}$ of some quantities, e.g., temperature, test scores, Dow Jones index, accidents, you name it. Meanwhile, for each observation b_i, we pair with some independent data $a_{i1}, a_{i2}, \ldots, a_{in}$, such as, humidity, age, unemployment rate, eyesight, etc.. An example: we observe that at 32 Fahrenheit (b_i, dependent variable), local humidity is 5%, local wind speed is 17 mph, and local pressure is 1 atm. In fact, we may draw up data in a table.

Temperature $\boldsymbol{b}$	Humidity $\boldsymbol{a}_1$	Wind Speed $\boldsymbol{a}_2$	Pressure $\boldsymbol{a}_3$	Air Quality Index $\boldsymbol{a}_4$
32	5	17	1	55
37	7	13	1.02	65
44	8	11	0.99	14
47	11	6	0.96	36
17	31	7	0.94	33
27	21	9	0.92	154

We seek the answer to the question: how does temperature depend on all these variables with the provided data? It is completely natural to posit a linear relationship between $\boldsymbol{b}$ and the $\boldsymbol{a}_i$'s. More precisely, for each b_i, we seek the coefficients $x_1, \ldots, x_n$ that

$$b_i = x_0 + x_1 a_{i1} + x_2 a_{i2} + \cdots + x_n a_n, \quad i = 1, 2, \ldots, m.$$

Is it utterly possible that we can find the exact $x_{i1}, \ldots, x_{in}$ that satisfy this relationship, **for every** i? A natural solvability condition is that we need n equations to determine these unknowns. However, in practice, we do not have control over the data we collect. There may be m data points for each quantity, while we can't just simply throw some out to allow this solvability because each data point contains some

truth of the underlying relationship. So, we seek coefficients $\{x_i\}_{i=1}^n$ that satisfy

$$\begin{aligned} b_1 &= x_0 + x_1 a_{11} + \cdots + x_n a_{1n} = (1, a_{11}, \ldots, a_{1n}) \cdot (x_0, x_1, \ldots, x_n) \\ &.. \\ &.. \\ b_m &= x_0 + x_1 a_{m1} + \cdots + x_n a_{mn} = (1, a_{m1}, \ldots, a_{mn}) \cdot (x_0, x_1, \ldots, x_n) \end{aligned}$$

or more compactly, noting from the inner product structure,

$$\boldsymbol{b} = \begin{bmatrix} 1 & a_{11} & a_{12} & . & . & a_{1n} \\ 1 & a_{21} & . & . & . & . \\ 1 & . & . & . & . & . \\ 1 & . & . & . & . & . \\ 1 & . & . & . & . & . \\ 1 & a_{m1} & . & . & . & a_{mn} \end{bmatrix}_{m \times (n+1)} \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix}_{(n+1) \times 1} = \boldsymbol{A}\boldsymbol{x}.$$

So, we have $(n+1)$ unknowns but only m equations. This is only possibly exactly solvable when $m = n+1$ which means we have exactly the same number of observation data points as the number of independent variables (and also that b lies in the column space of A). This is unrealistic. In practice, $m \gg n+1$, that is, we have massive amount of data, but only a handful of features we seek the extent of dependence on. The matrix A is typically "tall". Therefore, the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ here is not solvable in general.

Then what? Game over? No! We go for the next best thing. Now, if $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is not solvable, we may find some $\boldsymbol{x}$ that achieves $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$, which is a reasonable request. In fact, we seek a solution that minimizes the square of the l^2-error

$$\phi(\boldsymbol{x}) = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2$$

where $\phi : \mathbb{R}^{n+1} \to \mathbb{R}$. The vector $\boldsymbol{y} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ is known as the **residual vector**. All we are trying to do is to come up with a solution that minimizes the l^2-norm of this residual vector. This simple idea, in fact, has far reaching merits that go beyond linear algebra, and incidentally, beyond the scope of this class.

We expand the l^2-norm by definition,

$$\phi(\boldsymbol{x}) = \phi(x_0, x_1, \ldots, x_n) = \sum_{i=1}^{m} (b_i - x_0 - x_1 a_{i1} - x_2 a_{i2} - \cdots - x_n a_{in})^2 .$$

How do we minimize a function of multiple variables? We compute its gradient and

set it equal to $\mathbf{0}$ to find the critical points.

$$0 = \frac{\partial \phi}{\partial x_0} = -2 \sum_{i=1}^{m} \left(b_i - x_0 - x_1 a_{i1} - x_2 a_{i2} - \cdots - x_n a_{in} \right),$$
$$0 = \frac{\partial \phi}{\partial x_j} = -2 \sum_{i=1}^{m} \left(b_i - x_0 - x_1 a_{i1} - x_2 a_{i2} - \cdots - x_n a_{in} \right) a_{ij}, \quad j = 1, \ldots, n.$$

Moving the -2 out of the way, we see that the critical point(s) satisfy

$$0 = \frac{\partial \phi}{\partial x_0} = \sum_{i=1}^{m} \left(b_i - x_0 - x_1 a_{i1} - x_2 a_{i2} - \cdots - x_n a_{in} \right),$$
$$0 = \frac{\partial \phi}{\partial x_j} = \sum_{i=1}^{m} \left(b_i - x_0 - x_1 a_{i1} - x_2 a_{i2} - \cdots - x_n a_{in} \right) a_{ij}, \quad j = 1, \ldots, n.$$

Guess what? Now, we have exactly $n+1$ equations for the $n+1$ unknowns. This set of equations is called the **normal equations**.

The component of the residual vector is given by $y_i = b_i - x_0 - x_1 a_{i1} - x_2 a_{i2} - \cdots - x_n a_{in}$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)^{\mathrm{T}}$. Then, the equation reads

$$\sum_{i=1}^{m} y_i = 0,$$
$$\sum_{i=1}^{m} a_{ij} y_i = 0, \quad j = 1, \ldots, n.$$

which now requires you to recall the definition of matrix vector multiplication –

$$(\boldsymbol{A}\boldsymbol{x})_i = (a_{i1}, a_{i2}, \ldots, a_{in}) \cdot (x_1, x_2 \ldots, x_n) \implies i^{th} \text{ row of } \boldsymbol{A} \text{ dotted with } \boldsymbol{x}.$$

Let's visualize what $\sum_{i=1}^{m} a_{ij} y_i$ really is:

$$(a_{1j}, a_{2j}, a_{3j}, \ldots, a_{mj}) \cdot (y_1, \ldots, y_m)$$

where we realize that $(a_{1j}, a_{2j}, a_{3j}, \ldots, a_{mj})$ is the j^{th} **column** of $\boldsymbol{A}$, which means it is the j^{th} **row** of $\boldsymbol{A}^{\mathrm{T}}$. Thus,

$$\sum_{i=1}^{m} a_{ij} y_i = \sum_{i=1}^{m} a_{ji}^{\mathrm{T}} y_i = \left(\boldsymbol{A}^{\mathrm{T}} \boldsymbol{y} \right)_j, \quad j = 0, 1, \ldots, m.$$

Note that we have included the $j = 0$ case because

$$\sum_{i=1}^{m} y_i = (1, \ldots, 1) \cdot (y_1, \ldots, y_m)$$

uses precisely the first column of A (thus first row of A^{T}). Enumerating over all $j = 1, 2, \ldots, m$, we find that the **normal equations** can be written in matrix form,

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} = 0.$$

Now, looking at the definition of $\boldsymbol{y}$, we have

$$\begin{aligned} y_1 &= b_1 - x_0 - x_1 a_{11} - x_2 a_{12} - \cdots - x_n a_{1n} \\ &\vdots \\ y_m &= b_m - x_0 - x_1 a_{m1} - x_2 a_{m2} - \cdots - x_n a_{mn} \end{aligned}$$

which is

$$\boldsymbol{y} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}.$$

Inserting this back into the normal equation $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} = 0$, we have

$$\boldsymbol{A}^{\mathrm{T}}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\right) = 0 \implies \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{T}\boldsymbol{b},$$

the celebrated final form of the **normal equations**. All we need is the observation data: the dependent variable $\boldsymbol{b}$, and the independent variables $\boldsymbol{A}$. The matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is called the **Gramian** or **the Gram matrix**, named after Jørgen Pedersen Gram, a Danish actuary and mathematician.

We put the problem in full form: the (minimizer) solution to the **least square problem**

$$\widetilde{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2$$

is the solution to the set of **normal equations**

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\widetilde{\boldsymbol{x}} = \boldsymbol{A}^{T}\boldsymbol{b}.$$

Now, after finding where the critical point is, we still need to confirm that this critical point indeed gives me the minimum, not the maximum.

Theorem 9.1. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Every solution $\widetilde{\boldsymbol{x}}$ to $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\widetilde{\boldsymbol{x}} = \boldsymbol{A}^{T}\boldsymbol{b}$ satisfies*

$$\|\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}}\|_2 \leqslant \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2 \quad \forall \boldsymbol{x} \in \mathbb{R}^n,$$

that is, $\widetilde{\boldsymbol{x}}$, if exists, is the global minimizer of $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$.

Proof. Given $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$, we have

$$\|\boldsymbol{u} + \boldsymbol{v}\|_2^2 = (\boldsymbol{u} + \boldsymbol{v})^{\mathrm{T}}(\boldsymbol{u} + \boldsymbol{v}) = \|\boldsymbol{u}\|_2^2 + 2\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} + \|\boldsymbol{v}\|_2^2.$$

Then,

$$\begin{aligned}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 &= \|\boldsymbol{b}-\boldsymbol{A}\widetilde{\boldsymbol{x}}+\boldsymbol{A}\widetilde{\boldsymbol{x}}-\boldsymbol{A}\boldsymbol{x}\|_2^2 \\ &= \|\boldsymbol{b}-\boldsymbol{A}\widetilde{\boldsymbol{x}}\|_2^2 + 2\left(\boldsymbol{A}\left(\widetilde{\boldsymbol{x}}-\boldsymbol{x}\right)\right)^{\mathrm{T}}\left(\boldsymbol{b}-\boldsymbol{A}\widetilde{\boldsymbol{x}}\right) + \|\boldsymbol{A}\left(\widetilde{\boldsymbol{x}}-\boldsymbol{x}\right)\|_2^2 \\ &\geqslant \|\boldsymbol{b}-\boldsymbol{A}\widetilde{\boldsymbol{x}}\|_2^2 + 2\left(\widetilde{\boldsymbol{x}}-\boldsymbol{x}\right)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\cancelto{0}{\left(\boldsymbol{b}-\boldsymbol{A}\widetilde{\boldsymbol{x}}\right)} \\ &= \|\boldsymbol{b}-\boldsymbol{A}\widetilde{\boldsymbol{x}}\|_2^2 .\end{aligned}$$

□

9.2 Solution Technique

We use Cholesky decomposition on A^TA, contingent upon a verification that A^TA is symmetric and positive-definitsae. We first develop a lemma based on the rank-nullity theorem.

Lemma 9.1. *Given $A \in \mathbb{R}^{m\times n}$ (think of it as a linear map $A : \mathbb{R}^n \to \mathbb{R}^m$) the system $Ax = 0$ has the trivial solution $x = 0$ when $m > n$ (tall matrix) and all columns of A are linearly independent, i.e., rank$(A) = n$ (full rank).*

Proof. The proof relies on the rank-nullity theorem, a fundamental result in linear algebra. Suppose A is full rank. Then the rank is the minimum of m and n.

$$\begin{aligned}\dim(A) &= \operatorname{rank}(A) + \operatorname{nullity}(A) \\ n &= \min(m,n) + (\#\text{ solutions to } Ax = 0)\end{aligned}$$

where $\dim(A)$ is the size of the domain, namely, the number of variables, which has to be n to be dimensionally consistent for the equation $Ax = 0$.

If $n > m$, then $\min(m,n) = m$, and thus $(\#\text{ solutions to } Ax = 0) > 0$, implying that there is always nontrivial solutions to the system $Ax = 0$.

If $m > n$, then $\min(m,n) = n$, and thus $(\#\text{ solutions to } Ax = 0) = 0$, showing that the only solution to $Ax = 0$ is the trivial solution $x = 0$. □

Remark 9.1. *One should note that having linearly independent columns is sufficient, without mentioning $m > n$ since the matrix **cannot** have linearly independent columns if $n > m$ (think of having four vectors that live in $\mathbb{R}^2$).*

Theorem 9.2. *$A^{\mathrm{T}}A$ is positive definite if A has linearly independent columns.*

Proof. Consider v a nonzero column vector. Then,

$$v^{\mathrm{T}}\left(A^{\mathrm{T}}A\right)v = \left(v^{\mathrm{T}}A^{\mathrm{T}}\right)(Av) = (Av)^{\mathrm{T}}(Av) = \|Av\|_2^2 \geqslant 0.$$

To rid of the = sign, we use the lemma.

If A is a tall matrix (necessarily) with independent columns, then $Av = 0$ if and only if $v = 0$ by the lemma. But $v \neq 0$ by assumption. Thus, $A^{\mathrm{T}}A$ is positive definite. □

What does A having linearly independent mean in the context of the least square problem? Recall that A represents the data collected from the regressors, namely,

9.3 Optional: Existence of a Solution

It remains to show that $\widetilde{\boldsymbol{x}}$ indeed exists, and under one more condition on $\boldsymbol{A}$, is also unique. Existence is not hard if we know a little bit of linear algebra. Note that $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ lies in the range of $\boldsymbol{A}^{\mathrm{T}}$. But we also can show that the range of $\boldsymbol{A}^{\mathrm{T}}$ and that of $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ are the same (a fundamental theorem in linear algebra), which means there exists $\boldsymbol{x}$ such that $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ since both sides of the equation maps to the same subspace.

9.4 Optional: Uniqueness of the Solution

If $\det\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right) \neq 0$, then we are all set because then $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is invertible, and

$$\widetilde{\boldsymbol{x}} = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}.$$

But is this always the case? This should depend on $\boldsymbol{A}$ – but here $\boldsymbol{A}$ is not necessarily square. So the usual technique from linear algebra won't work.

The claim here is that $\boldsymbol{A}$ must have linearly independent columns iff $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is invertible. For the forward direction, assume that $\boldsymbol{A}$ has linearly independent columns, we suppose, on the contrary, that $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is not invertible. Then, $\det\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right) = 0$, which means there exists nonzero $\boldsymbol{z} = \boldsymbol{0}$ such that

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{z} = 0.$$

Now, multiplying $\boldsymbol{z}^{\mathrm{T}}$ on the left, we have

$$\boldsymbol{z}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{z} = 0 \implies (\boldsymbol{A}\boldsymbol{z})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{z}) = 0 \implies \|\boldsymbol{A}\boldsymbol{z}\|_2^2 = 0 \implies \boldsymbol{A}\boldsymbol{z} = 0, \quad \text{where } \boldsymbol{z} \neq \boldsymbol{0}.$$

But this is impossible because $\boldsymbol{A}$ has linearly independent columns, i.e., the only solution to $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{0}$ is $\boldsymbol{z} = \boldsymbol{0}$. Contradiction!

Part III
Iterative Methods for Linear Systems

In the previous section, we have studied direct methods of solving linear systems, in the sense that the error in our numerical solution arises purely from round-off errors. In this section, we study iterative methods, namely, approximating the true solution closer and closer, but only get close enough with a prescribed tolerance level.

10 Lecture X: Vector and Matrix Norms

Example 10.1. Suppose an iterative algorithm produces $x^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)$ as an approximate solution to a linear system at the k^{th} iteration.

If we know the true solution is x, we want to see how close we are. We need a measure of the "difference" $\left(x^{(k)} - x\right)$.

If we don't know the true solution (almost always), we want to see how much we have improved from the previous step, that is, we want to look at $\left(x^{(k)} - x^{(k-1)}\right)$ and see how this "difference" is changing as k changes. Certainly, if this "difference" becomes smaller and smaller, and if we can also prove (in mathematical analysis) that this "difference" goes to 0 as $k \to \infty$, then we know the sequence is **convergent**. You may read more about this on Cauchy sequence and its convergence given completeness of space.

10.1 Vector Norms

Definition 10.1. *A **vector norm** on $\mathbb{R}^n$ is a function, $\|\cdot\|$, from $\mathbb{R}^n$ to $\mathbb{R}$ with the following properties:*

1. *(nonnegativity) $\|\boldsymbol{x}\| \geqslant 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$,*
2. *(positive definiteness/point-separating) $\|\boldsymbol{x}\| = 0$ if and only if $\boldsymbol{x} = \mathbf{0}$,*
3. *(absolute homogeneity) $\|\alpha\boldsymbol{x}\| = |\alpha| \, \|\boldsymbol{x}\|$ for all $\alpha \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^n$,*
4. *(triangle inequality/subadditivity)$\|\boldsymbol{x} + \boldsymbol{y}\| \leqslant \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.*

We have a convention that a vector in $\mathbb{R}^n$ is a **column vector**. Then a **row vector** may be represented by the **transpose** of a vector, that is,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_5 \end{bmatrix} = (x_1, \ldots, x_5)^{\mathrm{T}}$$

where transpose simply means switching the dimension of the array.

Definition 10.2. *The l^2 and l^∞ norms for the vector $\boldsymbol{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$ are defined by*

$$\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}, \quad \|\boldsymbol{x}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i| .$$

The l^2-norm is also called **Euclidean norm** because it represents the usual notion of distance from the origin. You may deduce that the vectors that satisfies $\|\boldsymbol{x}\|_2 \leqslant 1$ cover a circle of radius 1 ($x_1^2 + x_2^2 \leqslant 1$), in 2D; or a sphere of radius 1 ($x_1^2 + x_2^2 + x_3^2 \leqslant 1$), in 3D. One should also note that the dot product of a vector to itself is

$$\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x} = \sum_{i=1}^{n} x_i^2 = \|\boldsymbol{x}\|_2^2 .$$

The l^∞-norm also has a geometric feature. It represents squares in 2D and cubes in 3D.

Example 10.2. Compute the l^2 and l^∞-norm of the vector $\boldsymbol{x} = (-1, 1, -2)^{\mathrm{T}}$.

$$\|\boldsymbol{x}\|_2 = \sqrt{(-1)^2 + 1^2 + (-2)^2} = \sqrt{6},$$

and

$$\|\boldsymbol{x}\|_\infty = \max \{|-1|, |1|, |-2|\} = 2.$$

In the definition of l^2 and l^∞-norm, we haven't really proved that they are indeed norms, that is, they satisfy the four properties given in the first definition. In fact, showing that l^∞ is a norm is not hard.

Proposition 10.1. *l^∞ is a norm.*

Proof. We check the four criteria.

1. (nonnegativitiy) For $\boldsymbol{x} \in \mathbb{R}^n$,

$$\|\boldsymbol{x}\|_\infty = \max_{1 \leqslant i \leqslant n} \{|x_i|\} \geqslant 0$$

since the absolute value is always nonnegative.

2. (positive definiteness) For $\boldsymbol{x} = \boldsymbol{0}$, then clearly, $\|\boldsymbol{x}\|_\infty = \|\boldsymbol{0}\|_\infty = \max_{1 \leqslant i \leqslant n} \{|0|\} = 0$. Suppose now $\|\boldsymbol{x}\|_\infty = 0$, then $\max_{1 \leqslant i \leqslant n} \{|x_i|\} = 0$, which implies $x_i = 0$ for all $i = 1, \ldots, n$.

3. (absolute homogeneity) For $\boldsymbol{x} \in \mathbb{R}^n$,

$$\|\alpha \boldsymbol{x}\|_\infty = \max_{1 \leqslant i \leqslant n} \{|\alpha x_i|\} = |\alpha| \max_{1 \leqslant i \leqslant n} \{|x_i|\} = |\alpha| \|\boldsymbol{x}\|_\infty .$$

4. (triangle inequality)

$$\begin{aligned} \|\boldsymbol{x} + \boldsymbol{y}\|_\infty &= \max_{1 \leqslant i \leqslant n} \{|x_i + y_i|\} \\ &\leqslant \max_{1 \leqslant i \leqslant n} (|x_i| + |y_i|) \\ &\leqslant \max_{1 \leqslant i \leqslant n} |x_i| + \max_{1 \leqslant i \leqslant n} |y_i| \\ &= \|\boldsymbol{x}\|_\infty + \|\boldsymbol{y}\|_\infty \end{aligned}$$

□

10.2 l^2 and l^∞ Vector Norms

However, to show that the l^2-norm is indeed a norm, we must use the following famous inequality.

Theorem 10.1. *(Cauchy-Schwarz Inequality) For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, we have*

$$\boldsymbol{x}^{\mathrm{T}} \boldsymbol{y} = \sum_{i=1}^n x_i y_i \leqslant \left\{ \sum_{i=1}^n x_i^2 \right\}^{1/2} \left\{ \sum_{i=1}^n y_i^2 \right\}^{1/2} = \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2 .$$

Proof. We have nothing to prove if $\boldsymbol{x} = \boldsymbol{0}$ or $\boldsymbol{y} = \boldsymbol{0}$. So, we suppose both vectors are nonzero.

Consider the scaled version of the two vectors, $\boldsymbol{u} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$ and $\boldsymbol{v} = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2}$. The reason why we consider these is because the original statement is equivalent to show that

$$\frac{\boldsymbol{x}^{\mathrm{T}} \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2} \leqslant 1.$$

So, it suffices to show that $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} \leqslant 1$.

Then, knowing that the dot product of a vector to itself is always nonnegative (yields the 2-norm squared, in fact), we have

$$\begin{aligned} 0 &\leqslant (\boldsymbol{u}-\boldsymbol{v})^{\mathrm{T}}(\boldsymbol{u}-\boldsymbol{v}) \\ &= \|\boldsymbol{u}\|_2^2 - 2\boldsymbol{u}^T\boldsymbol{v} + \|\boldsymbol{v}\|_2^2 \\ &= 2\left(1-\boldsymbol{u}^T\boldsymbol{v}\right) \end{aligned}$$

This implies

$$1-\boldsymbol{u}^T\boldsymbol{v} \geqslant 0 \implies \boldsymbol{u}^T\boldsymbol{v} \leqslant 1$$

Substituting $\boldsymbol{x}$ and $\boldsymbol{y}$ back, we have

$$\left(\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}\right)^{\mathrm{T}} \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2} \leqslant 1$$

and thus

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} \leqslant \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2 .$$

□

With Cauchy-Schwarz, we can now prove that $\|\cdot\|_2$ is indeed a norm. The only hard part is the triangle-inequality. For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, we have

$$\begin{aligned} \|\boldsymbol{x}+\boldsymbol{y}\|_2^2 &= (\boldsymbol{x}+\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{x}+\boldsymbol{y}) = \|\boldsymbol{x}\|_2^2 + 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + \|\boldsymbol{y}\|_2^2 \\ &\overset{\text{Cauchy-Schwarz}}{\leqslant} \|\boldsymbol{x}\|_2^2 + 2\|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2 + \|\boldsymbol{y}\|_2^2 = \left(\|\boldsymbol{x}\|_2 + \|\boldsymbol{y}\|_2\right)^2 . \end{aligned}$$

Taking a square root of both sides, we obtain

$$\|\boldsymbol{x}+\boldsymbol{y}\|_2 \leqslant \|\boldsymbol{x}\|_2 + \|\boldsymbol{y}\|_2 .$$

11 Lecture XI: Convergence with Respect to Norms and Norm Equivalence

11.1 Convergence

Now, knowing that $\|\cdot\|_2$ is a norm, we can go on measuring the difference between two vectors in $\mathbb{R}^n$.

Definition 11.1. *If* $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ *amd* $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^{\mathrm{T}}$ *are vectors in* $\mathbb{R}^n$*, and the* l^2 *and* l^∞ *distances between* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are defined by*

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \left(\sum_{i=1}^{n} (x_i - y_i)^2 \right)^{1/2}$$

and

$$\|\boldsymbol{x} - \boldsymbol{y}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i - y_i| .$$

Example 11.1. Suppose an approximate solution to a linear system, using 5-digit rounding, is

$$\tilde{\boldsymbol{x}} = (1.2001, 0.99991, 0.92538)^{\mathrm{T}}$$

and the true solution is

$$\boldsymbol{x} = (1, 1, 1)^{\mathrm{T}} .$$

Let's find the difference between them under the two settings of norm.

$$\begin{aligned} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_\infty &= \max \{|1 - 1.2001|, |1 - 0.99991|, |1 - 0.92538|\} \\ &= \max \{0.2001, 0.00009, 0.07462\} \\ &= 0.2001, \end{aligned}$$

and

$$\begin{aligned} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2 &= \left[(1 - 1.2001)^2 + (1 - 0.99991)^2 + (1 - 0.92538)^2 \right]^{1/2} \\ &= 0.21356. \end{aligned}$$

From this, we see that, even though $\widetilde{x_2}$ and $\widetilde{x_3}$ are good approximations, $\widetilde{x_1}$ really drags down the l^∞ error.

The concept of distance in $\mathbb{R}^n$ is also used to define a limit of sequence of vectors in this space.

Definition 11.2. *A sequence* $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{\infty}$ *of vectors in* $\mathbb{R}^n$ *is said to* ***converge*** *to* $\boldsymbol{x}$ *with respect to the norm* $\|\cdot\|$ *if, given any* $\epsilon > 0$*, there exists an integer* $N(\epsilon)$ *such that*

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\| < \epsilon, \quad \text{for all } k \geqslant N(\epsilon).$$

Remark 11.1. *One should think this sequence getting very close to* $\boldsymbol{x}$ *past certain index,*

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}, \boldsymbol{x}^{(N+1)}, \ldots$$

The statement is saying, given your tolerance level ϵ *between the approximant* $\boldsymbol{x}^{(k)}$ *and the target* $\boldsymbol{x}$*, one can always find an exact index* $N(\epsilon)$ *such that this distance between the approximant and the target is within* ϵ*. A convergent sequence has this property.*

On the other hand, a divergent sequence does not have this property. We state the negation of convergence. We say $\boldsymbol{x}^{(k)} \in \mathbb{R}^n$ *diverges with respect to the norm* $\|\cdot\|$ *if for every* $\boldsymbol{x} \in \mathbb{R}^n$*, there exists* $\epsilon > 0$ *such that for every integer* N*, there exists an index* $k \geqslant N$ *such that*

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\| \geqslant \epsilon.$$

You may consider the simple sequence $\boldsymbol{x}^{(k)} = (k, 0) \in \mathbb{R}^2$*. Eventually,* k *gets so big, say, to* N*, such that for every* $\boldsymbol{x} \in \mathbb{R}^2$*, one can look just a few more terms down the sequence to find* $\left\|\boldsymbol{x}^{(N+j)} - \boldsymbol{x}\right\| \geqslant \epsilon$ *where you are free to set* ϵ*. You can always find something larger to beat whatever tolerance you set, meaning that this sequence is indeed growing out of control.*

11.2 Equivalence of Norms on $\mathbb{R}^n$

Next, we discuss a result for l^∞, which then can be easily extended to other norms on $\mathbb{R}^n$ via another theorem (the equivalence theorem).

Theorem 11.1. *The sequence of vectors* $\left\{\boldsymbol{x}^{(k)}\right\}$ *converges to* $\boldsymbol{x}$ *in* $\mathbb{R}^n$ *with respect to the* l^∞*-norm if and only if* $\lim_{k\to\infty} x_i^{(k)} = x_i$*, for each* $i = 1, 2, \ldots, n$*.*

Proof. When proving statements involving if and only if such as $A \iff B$, we need to prove both directions of the statement, that is, $A \implies B$ and $B \implies A$.

1. $\implies$: assume that $\boldsymbol{x}^{(k)} \to \boldsymbol{x}$ in l^∞, prove that $\lim_{k\to\infty} x_i^{(k)} = x_i$, for each $i = 1, 2, \ldots, n$.

   Proof. Given $\epsilon > 0$, there exists an integer $N(\epsilon)$ such that for all $k \geqslant N(\epsilon)$,

   $$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\|_\infty = \max_{i=1,2,\ldots,n} \left|x_i^{(k)} - x_i\right| < \epsilon.$$

This result implies that $\left|x_i^{(k)} - x_i\right| < \epsilon$, for each $i = 1, 2, \ldots, n$, so $x_i^{(k)} \to x_i$ for each i. □

2. $\Longleftarrow$: assume that $\lim_{k\to\infty} x_i^{(k)} = x_i$, for each $i = 1, 2, \ldots, n$, prove that $\boldsymbol{x}^{(k)} \to \boldsymbol{x}$ in l^∞.

Proof. Given $\epsilon > 0$, there exists an integer $N_i(\epsilon)$ such that for all $k \geqslant N_i(\epsilon)$,

$$\left|x_i^{(k)} - x_i\right| < \epsilon,$$

for each $i = 1, 2, \ldots, n$.

Define $N(\epsilon) = \max_{i=1,2,\ldots,n} N_i(\epsilon)$. If $k \geqslant N(\epsilon)$, then

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\|_\infty = \max_{i=1,2,\ldots,n} \left|x_i^{(k)} - x_i\right| < \epsilon$$

which implies $\left\{\boldsymbol{x}^{(k)}\right\}$ converges to $\boldsymbol{x}$ with respect to the l^∞-norm. □

□

Remark 11.2. *You may wonder if this theorem only applies to the l^∞-norm. The next theorem will tell us the answer.*

Theorem 11.2. *For $\boldsymbol{x} \in \mathbb{R}^n$,*

$$\|\boldsymbol{x}\|_\infty \leqslant \|\boldsymbol{x}\|_2 \leqslant \sqrt{n}\,\|\boldsymbol{x}\|_\infty .$$

Proof. See Appendix C.9 in Bornemann, and a good proof by Steven Johnson. □

Remark 11.3. *This theorem also tells us that all norms on $\mathbb{R}^n$ are equivalent with respect to convergence. That is, if $\|\cdot\|$ and $\|\cdot\|'$ are any two norms on $\mathbb{R}^n$, and $\left\{\boldsymbol{x}^{(k)}\right\}$ has the limit $\boldsymbol{x}$ with respect to $\|\cdot\|$, then $\left\{\boldsymbol{x}^{(k)}\right\}$ also has the **same** limit $\boldsymbol{x}$ with respect to $\|\cdot\|'$.*

Remark 11.4. *To connect with the previous theorem that component-wise convergence and convergence are always true together, here we simply see that Theorem 1 is true for all l^p-norms in $\mathbb{R}^n$ (the converse of Theorem 2 becomes untrue when the vectors live in infinite-dimensional space).*

Remark 11.5. *This theorem is only valid for vectors in finite dimensional spaces. It is NOT valid if we look at infinite dimensional spaces, e.g., function spaces.*

Example 11.2. Consider the sequence

$$\boldsymbol{x}^{(k)} = \left(1, 2 + \frac{1}{k}, \frac{3}{k^3}, e^{-k}\sin k\right)^{\mathrm{T}} \in \mathbb{R}^4.$$

Find the limit of $\left\{\boldsymbol{x}^{(k)}\right\}$ in l^∞-norm and l^2-norm respectively.

Solution. l^∞-norm is straightforward. Theorem 1 tells us that the l^∞ limit of a sequence, if exists, is the limit of individual components. Thus,

$$\begin{aligned}
\lim_{k\to\infty} 1 &= 1;\\
\lim_{k\to\infty} 2 + \frac{1}{k} &= 2;\\
\lim_{k\to\infty} \frac{3}{k^2} &= 0;\\
\lim_{k\to\infty} e^{-k}\sin(k) &= 0,
\end{aligned}$$

and therefore, $\boldsymbol{x}^{(k)}$ converges to $\boldsymbol{x} = (1, 2, 0, 0)^{\mathrm{T}}$ with respect to the l^∞-norm.

To show that $\boldsymbol{x}^{(k)}$ converges to the same limit in l^2, we use Theorem 2 and the definition of convergence.

Since $\boldsymbol{x}^{(k)} \to \boldsymbol{x} = (1, 2, 0, 0)^{\mathrm{T}}$ in l^∞, for any $\epsilon > 0$, we can furnish an integer $N(\epsilon)$ such that

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\|_\infty < \frac{\epsilon}{2}$$

whenever $k \geqslant N(\epsilon)$. By Theorem 11.2, for exactly the same $N(\epsilon)$, we have

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\|_2 \overset{\text{Theorem 2 for } n=4}{\leqslant} \sqrt{4}\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\right\|_\infty \leqslant 2\left(\frac{\epsilon}{2}\right) = \epsilon$$

for $k \geqslant N(\epsilon)$. Thus $\boldsymbol{x}^{(k)}$ also converges to $\boldsymbol{x}$ in l^2-norm.

Example 11.3. Sublevel sets of $\|\boldsymbol{x}\|_\infty$ and $\|\boldsymbol{x}\|_2$ for $\boldsymbol{x} \in \mathbb{R}^2$.

Consider $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_\infty \leqslant 1$. This function outputs a square with sidelength 2, centered at $(0, 0)$.

$g(\boldsymbol{x}) = \|\boldsymbol{x}\|_2 \leqslant 1$ covers the circle of radius 1.

12 Lecture XII: Matrix Norms

12.1 Definition of a Matrix Norm

We have now seen a measure of distance between vectors in $\mathbb{R}^n$. Is there such a thing for matrices, namely, arrays that lives in $\mathbb{R}^{n\times n}$ (square matrices)? And even if there is, what geometric meaning does it have? We can surely visualize l^2 or l^∞ norms of vectors, but what about the "norm" of a matrix?

Definition 12.1. *A **matrix norm** on the set of all $n \times n$ matrices is a real-valued function $\|\cdot\|$, defined on this set, satisfying for all $n \times n$ matrices A and B and all real numbers α:*

1. $\|A\| \geqslant 0$;
2. $\|A\| = 0$ *if and only if A is the zero matrix;*
3. $\|\alpha A\| = |\alpha| \, \|A\|$;
4. $\|A + B\| \leqslant \|A\| + \|B\|$;
5. $\|AB\| \leqslant \|A\| \, \|B\|$.

Then, the distance between $n \times n$ matrices A and B with respect to this matrix norm is $\|A - B\|$.

The only additional requirement (compared to a vector norm) is the fifth item. Still, all of these criteria are very abstract. Can we utilize the notion of a vector norm to induce a norm on a matrix? If so, this then implies that a matrix norm can be constructed via a vector norm, which can be readily computed.

Theorem 12.1. *If $\|\cdot\|$ is a vector norm on $\mathbb{R}^n$, then*

$$\|A\| = \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\| = \max_{\boldsymbol{z}\neq\boldsymbol{0}} \frac{\|A\boldsymbol{z}\|}{\|\boldsymbol{z}\|}$$

is a matrix norm.

Proof. We have nothing to prove if A is the zero matrix. The first three criteria are

simple. We work with the first definition.

$$\begin{aligned}
\|A+B\| &= \max_{\|\boldsymbol{x}\|=1} \|(A+B)\,\boldsymbol{x}\| \\
&= \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}+B\boldsymbol{x}\| \\
&\overset{\text{triangle ineq. for vector norms}}{\leqslant} \max_{\|\boldsymbol{x}\|=1} \left(\|A\boldsymbol{x}\|+\|B\boldsymbol{x}\|\right) \\
&\leqslant \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\| + \max_{\|\boldsymbol{x}\|=1} \|B\boldsymbol{x}\| \\
&= \|A\|+\|B\|\,.
\end{aligned}$$

The last one requires some more work.

$$\|AB\| = \max_{\|\boldsymbol{x}\|=1} \|AB\boldsymbol{x}\|$$

and now we are stuck because we wish we can bound the $\|AB\boldsymbol{x}\|$ by something nice.

Lemma 12.1. $\|A\boldsymbol{x}\| \leqslant \|A\|\,\|\boldsymbol{x}\|$ *for* $\boldsymbol{x} \neq \mathbf{0}$.

Proof. Define $\boldsymbol{y} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$. Then,

$$\|A\boldsymbol{x}\| \overset{\text{absolute homogeneity}}{=} \left\|A\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\right\| \|\boldsymbol{x}\| \leqslant \max_{\|\boldsymbol{y}\|=1} \|A\boldsymbol{y}\|\,\|\boldsymbol{x}\| = \|A\|\,\|\boldsymbol{x}\|\,.$$

□

Now, we go on with the proof for the (submultiplicative) property:

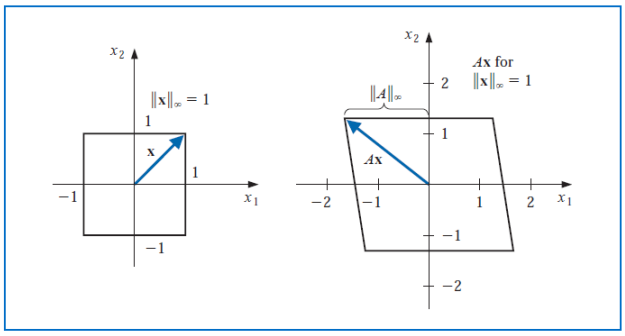
$$\begin{aligned}
\|AB\| &= \max_{\|\boldsymbol{x}\|=1} \|AB\boldsymbol{x}\| \\
&\leqslant \max_{\|\boldsymbol{x}\|=1} \|A\|\,\|B\boldsymbol{x}\| \\
&= \|A\| \max_{\|\boldsymbol{x}\|=1} \|B\boldsymbol{x}\| \\
&= \|A\|\,\|B\|\,.
\end{aligned}$$

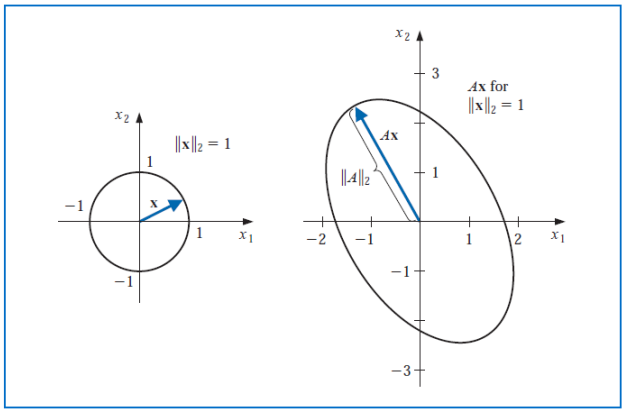
□

Any vector norm on $\mathbb{R}^n$ **induces** matrix norm. So, we can consider

$$\begin{aligned}
\|A\|_\infty &= \max_{\|\boldsymbol{x}\|_\infty=1} \|A\boldsymbol{x}\|_\infty\,, \quad l^\infty\text{-norm;} \\
\|A\|_2 &= \max_{\|\boldsymbol{x}\|_2=1} \|A\boldsymbol{x}\|_2\,, \quad l^2\text{-norm.}
\end{aligned}$$

But again, what is geometrically meaningful of these matrix norms?





We learn a great deal from the action of a matrix on a vector. In principle, $A\boldsymbol{x}$ simply changes the direction and magnitude of $\boldsymbol{x}$. Therefore, if we consider the vectors that satisfy $\|\boldsymbol{x}\|_2 = 1$, namely, the vectors that live on the unit circle, a matrix A will transform this circle to an ellipse, but stretching (or compressing) the length, while rotating the principle axes of the circle. The figure above showcases the matrix norm where

$$A = \begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix}.$$

In fact, this matrix rotates any initial vector $\boldsymbol{x}$ by 90 degrees counterclockwise, and then stretches twice as long.

12.2 Spectral Radius and $\|A\|_2$

Definition 12.2. *The **spectral radius** $\rho(A)$ of a matrix A defined by*

$$\rho(A) = \max|\lambda|$$

where λ is an eigenvalue of A. If λ is complex, i.e., $\lambda = \alpha + i\beta$, define

$$|\lambda| = \sqrt{\alpha^2 + \beta^2}.$$

As it turns out, the spectral radius can be linked to the matrix norm induced by l^2-norm and the Gramian matrix $A^{\mathrm{T}}A$.

Theorem 12.2. *If A is an $n \times n$ matrix, then*

1. $\|A\|_2 = \left[\rho\left(A^{\mathrm{T}}A\right)\right]^{1/2}$,
2. $\rho(A) \leqslant \|A\|$, *for any induced norm $\|\cdot\|$ (also known as natural norm).*

Let us use the first formula to compute the 2-norm of a matrix.

Example 12.1. Determine the l^2-norm of

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

Solution 12.1. We need $\rho\left(A^{\mathrm{T}}A\right)$. But first, we compute $A^{\mathrm{T}}A$.

$$A^{\mathrm{T}}A = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 2 & -1 \\ 2 & 6 & 4 \\ -1 & 4 & 5 \end{bmatrix} = B.$$

Then, we find the eigenvalues of B.

$$\begin{aligned} \det(B - \lambda I) &= \det \begin{bmatrix} 3-\lambda & 2 & -1 \\ 2 & 6-\lambda & 4 \\ -1 & 4 & 5-\lambda \end{bmatrix} \\ &= -\lambda^3 + 14\lambda^2 - 42\lambda \\ &= -\lambda\left(\lambda^2 - 14\lambda + 42\right). \end{aligned}$$

Thus, $\lambda = 0$, or $\lambda = 7 \pm \sqrt{7}$.

$$\|A\|_2 = \sqrt{\rho\left(A^{\mathrm{T}}A\right)} = \sqrt{\max\left\{0, 7-\sqrt{7}, 7+\sqrt{7}\right\}} = \sqrt{7+\sqrt{7}}.$$

12.3 $\|A\|_\infty$ norm

We first provide a formula for the l^∞-norm of a matrix, as it comes in quite handy when we need to compute any induced matrix norms.

Theorem 12.3. *Let A be an $n \times n$ square matrix. Then,*

$$\|A\|_\infty \overset{\textit{definition}}{=} \max_{\|\boldsymbol{x}\|\neq 0} \|A\boldsymbol{x}\| = \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|\,,$$

namely, the maximum row sum of absolute values of the entries.

Proof. See Theorem 7.11 in Burden and Faires. The proof involves establishing both directions of the equality, namely, $\|A\|_\infty \leqslant \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|$ and $\|A\|_\infty \geqslant \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|$. □

12.4 Condition Number of a Matrix

With a formula to compute $\|A\|_2$ via the spectral radius, we can then define the condition number of a matrix $\kappa(A)$. Before that, we consider a simple system of equations to showcase the importance of knowing the condition numbers, before we use the matrix to solve any problems.

Example 12.2. Consider the linear system $A\boldsymbol{x} = \boldsymbol{b}$ given by

$$\begin{bmatrix} 1 & 2 \\ 1.0001 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3.0001 \end{bmatrix}$$

has the uniques solution $\boldsymbol{x} = (1,1)^{\mathrm{T}}$. Determine the residual vector for the poor approximation $\widetilde{\boldsymbol{x}} = (3, -0.0001)^{\mathrm{T}}$, define as $\boldsymbol{r} = \boldsymbol{b} - A\widetilde{\boldsymbol{x}}$.

Solution. We have

$$\boldsymbol{r} = \boldsymbol{b} - A\widetilde{\boldsymbol{x}} = \begin{bmatrix} 3 \\ 3.0001 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 1.0001 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ -0.0001 \end{bmatrix} = \begin{bmatrix} 0.0002 \\ 0 \end{bmatrix}$$

and observe that $\|\boldsymbol{r}\|_\infty = 0.0002$, which is reasonably small. However, obviously the approximation is very poor,

$$\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|_\infty = 2.$$

In the previous example, we see $\|\boldsymbol{r}\|_\infty = \|A\boldsymbol{x} - A\widetilde{\boldsymbol{x}}\|_\infty$ as the "perturbation to input", while $\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|_\infty$ as perturbation to output. So, if we can somehow relate them, we have an estimate of the "condition number" of the matrix A.

Theorem 12.4. *Suppose $\widetilde{\boldsymbol{x}}$ is an approximation to the solution of $A\boldsymbol{x} = \boldsymbol{b}$, where A is nonsingular, and $\boldsymbol{r}$ is the residual vector for $\widetilde{\boldsymbol{x}}$. Then, for any induced norm, for $\boldsymbol{x} \neq 0$ and $\boldsymbol{b} \neq 0$, we have*

$$\frac{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \leqslant \|A\| \left\|A^{-1}\right\| \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}.$$

Remark 12.1. *We skip the proof (See Theorem 7.27 in Burden and Faires). However, this important result leads us to the definition of the condition number of a matrix $\kappa(A)$. Note that the LHS $\frac{\|\boldsymbol{x}-\widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|}$ is **relative error in output** (solution), and the RHS is*

$$\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|} = \frac{\|\boldsymbol{b} - A\widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{b}\|}$$

*can be viewed as a **relative error in input**. So, we may define the **relative condition number** of A as*

$$\kappa(A) = \|A\| \left\|A^{-1}\right\|.$$

By a property of the matrix norm (sub-multiplicativity), we have

$$\kappa(A) = \|A\| \left\|A^{-1}\right\| \geqslant \left\|AA^{-1}\right\| = \|I\| = 1.$$

*We say that a matrix is **well-condition** if $\kappa(A)$ is close to 1, and is **ill-conditioned** when $\kappa(A)$ is significantly greater than 1.*

Example 12.3. Determine the condition number of the matrix in the last example.

Solution. We are allowed to use any induced norm, so let's do the easiest one, $\|A\|_\infty$.

$$\|A\|_\infty = \max\{|1| + |2|, |1.0001| + |2|\} = 3.0001$$

which is not too big. However,

$$A^{-1} = \begin{bmatrix} -10000 & 10000 \\ 5000.5 & -5000 \end{bmatrix}$$

and thus

$$\left\|A^{-1}\right\|_\infty = 20000.$$

Altogether, $\kappa(A) \approx 60000$ which means the matrix is ill-conditioned (but I've seen worse).

13 Lecture XIII: Jacobi Iteration

For a linear system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned}$$

we can express each component of the solution $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ in terms of other components. More precisely, by isolating x_i in the i^{th} equation, we have

$$\begin{aligned} x_1 &= \frac{b_1 - (a_{12}x_2 + \cdots + a_{1n}x_n)}{a_{11}}, \\ x_2 &= \frac{b_2 - (a_{21}x_1 + a_{23}x_3 + \cdots + a_{2n}x_n)}{a_{22}}, \\ &\vdots \\ x_n &= \frac{b_n - \left(a_{n1}x_1 + \cdots + a_{(n-1)n}x_{n-1}\right)}{a_{nn}}. \end{aligned}$$

This looks a lot like the "naive substitution" technique introduced before Gaussian elimination. Indeed, this is a costly algorithm. However, some two hundred years ago, Carl Gustav Jacob Jacobi (also responsible for the Jacobian matrix for multidimensional change of variable) found that you can start with some guesses

$$\boldsymbol{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$$

and plug them into the RHS. We then obtain a new vector

$$\boldsymbol{x}^{(1)} = \left(x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)}\right).$$

Now, chuck this into the RHS, and continue iterating, we "somehow" could obtain the true solution $\boldsymbol{x}$ by approximating with

$$\boldsymbol{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right).$$

This iteration ends when the sequential error of the approximation is below some prescribed tolerance ϵ, i.e.,

$$\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\| < \epsilon$$

under various notions of vector norms (and also various notions of error). Other forms of stopping criterion may be

$$\frac{\left\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right\|}{\left\|\boldsymbol{x}^{(k)}\right\|} < \epsilon$$

which is well-defined because $\left\|\boldsymbol{x}^{(k)}\right\| \neq 0$ for $A\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{b} \neq \boldsymbol{0}$. The criterion can depend on the problem at hand.

In component form, this method can be written as

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\right), \quad i = 1, 2, \ldots, n.$$

13.1 A Worked Example

Example 13.1. $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$, $b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$. Let's solve $A\boldsymbol{x} = b$ using Jacobi iteration, with an initial guess $\boldsymbol{x}^{(0)} = (0, 0)^{\mathrm{T}}$. Note that the true solution is $\boldsymbol{x} = (1, 1)^{\mathrm{T}}$.

$$\begin{aligned} x_1^{(k+1)} &= \frac{b_1 - a_{12} x_2^{(k)}}{a_{11}} = \frac{3 - 1x_2^{(k)}}{2}, \\ x_2^{(k+1)} &= \frac{b_2 - a_{21} x_1^{(k)}}{a_{22}} = \frac{3 - 1x_1^{(k)}}{2}. \end{aligned}$$

With this formula, we have

$$\begin{aligned} \boldsymbol{x}^{(1)} &= \left(x_1^{(1)}, x_2^{(1)}\right)^{\mathrm{T}} = \left(\frac{3}{2}, \frac{3}{2}\right) \\ \boldsymbol{x}^{(2)} &= \left(x_1^{(2)}, x_2^{(2)}\right)^{\mathrm{T}} = \left(\frac{3 - \frac{3}{2}}{2}, \frac{3 - \frac{3}{2}}{2}\right) = \left(\frac{3}{4}, \frac{3}{4}\right) \\ \boldsymbol{x}^{(3)} &= \left(x_1^{(3)}, x_2^{(3)}\right)^{\mathrm{T}} = \left(\frac{3 - \frac{3}{4}}{2}, \frac{3 - \frac{3}{4}}{2}\right) = \left(\frac{9}{8}, \frac{9}{8}\right) \\ \boldsymbol{x}^{(4)} &= \left(x_1^{(4)}, x_2^{(4)}\right)^{\mathrm{T}} = \left(\frac{3 - \frac{9}{8}}{2}, \frac{3 - \frac{9}{8}}{2}\right) = \left(\frac{15}{16}, \frac{15}{16}\right) \end{aligned}$$

As one iterates further, one finds that the solution converges to $(1, 1)$.

But **why** does this algorithm converge? The construction seems to be a natural extension of the naive substitution, but its convergence is somewhat unguided. In fact, is convergence even guaranteed for any initial guesses? In terms of analysis in linear algebra, we should start by condensing the iterative algorithm into a functional or matrix form. In the case of Jacobi Iteration, we put it in matrix form first.

13.2 Matrix Form of Jacobi Iteration

Let us revisit the overall structure of the Jacobi method.

$$
\begin{aligned}
x_1 &= \frac{b_1 - (a_{12}x_2 + \cdots + a_{1n}x_n)}{a_{11}}, \\
x_2 &= \frac{b_2 - (a_{21}x_1 + a_{23}x_3 + \cdots + a_{2n}x_n)}{a_{22}}, \\
&\vdots \\
x_n &= \frac{b_n - \left(a_{n1}x_1 + \cdots + a_{(n-1)n}x_{n-1}\right)}{a_{nn}}.
\end{aligned}
$$

Here, we see that each row involves other variables. Rearranging a little, we see that

$$a_{11}x_1 = b_1 - (a_{12}x_2 + \cdots + a_{1n}x_n)$$

More precisely, let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$, we observe

$$a_{11}x_1 = b_1 - (0, a_{12}, a_{13}, \ldots, a_{1n})\,\boldsymbol{x}.$$

while the left hand side is no other than the diagonal element multiplying the corresponding unknown.

In general, writing all n equations in the form above, we have

$$
\begin{bmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & . & & \\ & & & . & \\ & & & & a_{nn} \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix}
=
\begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}
-
\begin{bmatrix} 0 & a_{12} & . & . & a_{1n} \\ a_{21} & 0 & . & . & a_{2n} \\ . & & 0 & & . \\ . & & & 0 & . \\ a_{n1} & & & & 0 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix}.
$$

Define the diagonal, upper triangular and lower triangular part of the matrix $A = D + L + U$,

$$
D = \begin{bmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & . & & \\ & & & . & \\ & & & & a_{nn} \end{bmatrix}, \quad
L = \begin{bmatrix} 0 & 0 & . & . & 0 \\ a_{21} & 0 & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{n1} & . & . & a_{n,n-1} & 0 \end{bmatrix}, \quad
U = \begin{bmatrix} 0 & a_{12} & . & . & a_{1n} \\ 0 & 0 & . & . & a_{2n} \\ . & . & 0 & . & . \\ . & . & . & 0 & a_{n-1,n} \\ 0 & . & . & . & 0 \end{bmatrix},
$$

we can rewrite the system as

$$D\boldsymbol{x} = \boldsymbol{b} - (L + U)\,\boldsymbol{x}.$$

Multiplying both sides of the equation by D^{-1}, which exists (just a diagonal matrix of the reciprocals of D), we have

$$\boldsymbol{x}^{(k+1)} = -D^{-1}\left(L+U\right)\boldsymbol{x}^{(k)} + D^{-1}b.$$

Note that in component form, this is exactly the same formula as we had in the last section,

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, n,$$

since $D^{-1} = \left\{\frac{1}{a_{ii}}\right\}_{i=1}^{n}$.

13.3 Gauss-Seidel Iteration

13.3.1 Jacobi Method Revisited

Let's revisit Jacobi's method,

$$\begin{aligned}
x_1^{(k+1)} &= \frac{b_1 - \left(a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)}\right)}{a_{11}},\\
x_2^{(k+1)} &= \frac{b_2 - \left(a_{21}x_1^{(k)} + a_{23}x_3^{(k)} + \cdots + a_{2n}x_n^{(k)}\right)}{a_{22}},\\
&\vdots\\
x_n^{(k+1)} &= \frac{b_n - \left(a_{n1}x_1^{(k)} + \cdots + a_{(n-1)n}x_{n-1}^{(k)}\right)}{a_{nn}}.
\end{aligned}$$

Consider the first component of the iterates, $x_1^{(k+1)}$. It only depends on $x_2^{(k)}, \ldots, x_n^{(k)}$ but not $x_1^{(k)}$. So, we make an independent update $x_1^{(k+1)}$. Then for $x_2^{(k+1)}$, we are still using the **old** value $x_1^{(k)}$, even though we have just computed the latest update $x_1^{(k+1)}$. **Gauss-Seidel method utilizes this brilliant realisation.** Since the Jacobi method sequentially computes the iterates by each component, we can utilize the immediate information obtained within a single step of the algorithm to help us advance the sequence. In other words, if we know $x_1^{(k+1)}$ is a better candidate to help us determine $x_2^{(k+1)}$ than $x_1^{(k)}$, we should use it by all means. This means $x_3^{(k+1)}$ will use the immediate updates of $x_1^{(k+1)}$ and $x_2^{(k+1)}$ instead of $x_1^{(k)}$ and $x_2^{(k)}$. So on and so forth.

How do we write this out in a more compact form? First, we list the actual equations. Let's consider a 3×3 matrix as an example of a more general approach.

Example 13.2. Consider the Gauss-Seidel iterative method for a 3×3 system,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1; \\ a_{21}x_1 + a_{22}x_2 + a_{13}x_3 &= b_2; \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3. \end{aligned}$$

Gauss-Seidel method surmises a **component formulation**,

$$\begin{aligned} x_1^{(k+1)} &= \frac{b_1 - \left(a_{12}x_2^{(k)} + a_{13}x_3^{(k)}\right)}{a_{11}}, \\ x_2^{(k+1)} &= \frac{b_2 - \left(\boxed{a_{21}x_1^{(k+1)}} + a_{23}x_2^{(k)}\right)}{a_{22}}, \\ x_3^{(k+1)} &= \frac{b_3 - \left(\boxed{a_{31}x_1^{(k+1)}} + \boxed{a_{32}x_2^{(k+1)}}\right)}{a_{33}}, \end{aligned}$$

where we highlight the boxed terms: they are the immediate updates from the previous lines, within the same k^{th} step of the algorithm.

Now, let's put the $(k+1)^{th}$ iterates on one side, and k^{th} ones on the other – that is, move the boxes to the LHS. We have

$$\begin{aligned} a_{11}x_1^{(k+1)} &= b_1 - \left(a_{12}x_2^{(k)} + a_{13}x_3^{(k)}\right), \\ a_{21}x_1^{(k+1)} + \boxed{a_{22}x_2^{(k+1)}} &= b_2 - a_{23}x_2^{(k)}, \\ \boxed{a_{31}x_1^{(k+1)}} + \boxed{a_{32}x_2^{(k+1)}} + a_{33}x_3^{(k+1)} &= b_3. \end{aligned}$$

How shall we write the LHS as a matrix-vector multiplication? The vector is obviously the update $\boldsymbol{x}^{(k+1)}$. The matrix, in fact, is the lower triangular section of A. We see it this way:

$$\begin{aligned} a_{11}x_1^{(k+1)} + 0x_2^{(k+1)} + 0x_3^{(k+1)} &= b_1 - \left(0x_1^{(k)} + a_{12}x_2^{(k)} + a_{13}x_3^{(k)}\right), \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} + 0x_3^{(k+1)} &= b_2 - \left(0x_1^{(k)} + 0x_2^{(k)} + a_{23}x_2^{(k)}\right), \\ a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} &= b_3 - \left(0x_1^{(k)} + 0x_2^{(k)} + 0x_3^{(k)}\right). \end{aligned}$$

More concretely, the Gauss-Seidel method has the following **matrix formulation**

$$LHS = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix}$$
$$= (D + L)\, \boldsymbol{x}^{(k+1)}$$

In the RHS, we see that the data vector $\boldsymbol{b}$ stays put, while the rest of the coefficients form the upper triangular part of A, namely,

$$RHS = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} - \begin{bmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} = \boldsymbol{b} - U\boldsymbol{x}^{(k)}.$$

13.3.2 Matrix and Component Form of Gauss-Seidel

In general, the Gauss-Seidel method, in **matrix formulation**, reads

$$(D + L)\, \boldsymbol{x}^{(k+1)} = -U\boldsymbol{x}^{(k)} + \boldsymbol{b}.$$

Taking an inverse of $D + L$, we have

$$\text{(Gauss-Seidel)} \quad \boldsymbol{x}^{(k+1)} = -\left(D + L\right)^{-1} U\boldsymbol{x}^{(k)} + \left(D + L\right)^{-1} \boldsymbol{b},$$

where we identify the **Iterative Linear Operator** for the Gauss-Seidel method is

$$\boxed{T_{\text{GS}} = -\left(D + L\right)^{-1} U}$$

In component form, for $i = 1, 2, \ldots, n$, we have

$$\boxed{x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right]},$$

by considering the individual equations. To compute the update for the $x_i^{(k+1)}$, we need to use $\left\{ x_j^{(k+1)} \right\}_{j=1}^{i-1}$, that is, the immediate updates within the same step for all previous unknowns $x_1^{(k+1)}, \ldots x_{i-1}^{(k+1)}$, AND the ones we haven't updated, $\left\{ x_j^{(k)} \right\}_{j=i+1}^{n}$.

13.3.3 A Worked Example using Gauss-Seidel

Let's consider the same system as the one in Jacobi iteration, $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$, with an initial guess $\boldsymbol{x}^{(0)} = (0, 0)$. Gauss-Seidel method says

$$x_1^{(1)} = \frac{b_1 - a_{12}x_2^{(0)}}{a_{11}} = \frac{3 - 1 \times 0}{2} = \boxed{\frac{3}{2}},$$
$$x_2^{(1)} = \frac{b_1 - a_{21}\boxed{x_1^{(1)}}}{a_{22}} = \frac{3 - 1 \times \boxed{\frac{3}{2}}}{2} = \frac{3}{4}.$$

We have the next iterates

$$x_1^{(2)} = \frac{3 - 1 \times \frac{3}{4}}{2} = \boxed{\frac{9}{8}}, \quad x_2^{(2)} = \frac{3 - 1 \times \boxed{\frac{9}{8}}}{2} = \frac{15}{16},$$

and

$$x_1^{(3)} = \frac{3 - 1 \times \frac{15}{16}}{2} = \boxed{\frac{33}{32}}, \quad x_2^{(3)} = \frac{3 - 1 \times \boxed{\frac{33}{32}}}{2} = \frac{63}{64}$$

where the boxed terms highlight the immediate usage of a previous update.

13.3.4 Comparison

Compare the Gauss-Seidel method to the Jacobi method,

$$\text{(Gauss-Seidel)} \quad \boldsymbol{x}^{(k+1)} = -\left(D + L\right)^{-1} U\boldsymbol{x}^{(k)} + \left(D + L\right)^{-1}\boldsymbol{b}.$$

$$\text{(Jacobi)} \quad \boldsymbol{x}^{(k+1)} = -D^{-1}\left(L + U\right)\boldsymbol{x}^{(k)} + D^{-1}\boldsymbol{b}.$$

Both methods are in the form of

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + c$$

where T, the **Iterative Linear Operator**, is different for the methods applied, i.e.

$$T_J = -D^{-1}\left(L + U\right),$$
$$T_{GS} = -\left(D + L\right)^{-1} U.$$

13.3.5 A Geometric Interpretation of Gauss-Seidel and its Extensions

Take $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$. Perform Gauss-Seidel on $A\boldsymbol{x} = b$, knowing that the true solution $\boldsymbol{x} = (1,1)^{\mathrm{T}}$. We have the iterates

$$\begin{aligned} \boldsymbol{x}^{(0)} &= (0,0)^{\mathrm{T}} \\ \boldsymbol{x}^{(1)} &= \left(\frac{3}{2}, \frac{3}{4}\right)^{\mathrm{T}} \\ \boldsymbol{x}^{(2)} &= \left(\frac{9}{8}, \frac{15}{16}\right)^{\mathrm{T}} \end{aligned}$$

Plotting these points in the x_1-x_2 plane, we find that when we travel from $\boldsymbol{x}^{(1)}$ to $\boldsymbol{x}^{(2)}$, we wish we go a little harder so that we actually hit the true solution in this direction. In fact, this direction is characterized by the vector $\boldsymbol{x}^{(2)} - \boldsymbol{x}^{(1)}$.

Please consult the Lecture 14 slides for a picture presentation of this iteration in the x_1-x_2-plane.

We now want to control how "hard" we move in this direction at every step of the Gauss-Seidel iteration, namely, put a control on $\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$ so that we move further than Gauss-Seidel intends to.

13.4 Sufficient Condition for Convergence of Jacobi and Gauss-Seidel

13.4.1 Sufficient Condition for Convergence of Jacobi

The matrix formulation of Jacobi's iteration satisfies

$$\boldsymbol{x}^{(k+1)} = -D^{-1}(L+U)\boldsymbol{x}^{(k)} + D^{-1}\boldsymbol{b} = T_J\boldsymbol{x}^{(k)} + c$$

where

$$T_J = -D^{-1}(L+U)$$

is called the **Iterative Linear Operator** for the Jacobi Iteration; $c = D^{-1}\boldsymbol{b}$ is a constant, at each step of the algorithm.

Now, we want to see if the error between successive approximations is getting smaller or not. We first compute

$$\begin{aligned} e^{(k+1)} = \left\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right\| &= \left\|T_J\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right)\right\| \\ &= \left\|T_J\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right)\right\| \\ &\overset{\text{Lemma 12.1}}{\leqslant} \|T_J\| \left\|\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right)\right\| \\ &= \left\|D^{-1}(L+U)\right\| e^{(k)} \end{aligned}$$

So, we see that if $\|D^{-1}(L+U)\| < 1$, then the sequential error will decrease and converge to zero (true for any matrix norm $\|\cdot\|$).

Is there anything special about matrices $A = D+L+U$ that satisfies $\|D^{-1}(L+U)\| < 1$? Let's look at what $D^{-1}(L+U)$ really means. Suppose we are looking at the l^∞-norm. Then, by Theorem 12.3 (induced definition of the matrix infinity norm)

$$\left\|D^{-1}(L+U)\right\|_\infty = \max_{1\leqslant i\leqslant n} \frac{1}{|a_{ii}|} \sum_{j\neq i}^{n} |a_{ij}|.$$

Requiring that $\|D^{-1}(L+U)\|_\infty < 1$ means that

$$\max_{1\leqslant i\leqslant n} \frac{1}{|a_{ii}|} \sum_{j\neq i}^{n} |a_{ij}| < 1 \implies \sum_{j\neq i}^{n} |a_{ij}| < |a_{ii}|, \text{ for each (row) } i = 1, 2, \ldots, n.$$

This is equivalent to saying that the diagonal entry must be very large, at least as large as the absolute sum of the rest of the entries in the same row. This type of matrices is call **strictly diagonally dominant matrix** (if the inequality is $\leqslant$, then it is just **diagonally dominant**). Jacobi method is **guaranteed** to work for these matrices.

You may wonder, what if my matrix is not diagonally dominant, does the Jacobi method still work? The answer is, sometimes. There is in fact a much more subtle condition called Sassenfeld condition (uploaded on Canvas) that is less stringent than $\|D^{-1}(L+U)\| < 1$. It is a milder set of sufficient conditions that include a wider range of matrices that ensures the convergence of the Jacobi iteration.

13.4.2 (Optional) Sufficient Condition for Convergence of Gauss-Seidel

Can we employ a similar argument to derive the sufficient condtion for Gauss-Seidel to converge? We work with the iteration matrix

$$T_{GS} = -(D+L)^{-1} U.$$

However, forcing $||T_{GS} < 1||$ no longer gives an entry-wise convenient express as Jacobi did. So, it seems obtaining a clearcut sufficient conditions based on the entries of A for the convergence of Gauss-Seidel seems futile.

One question remains: if A is diagonally dominant, will the Gauss-Seidel method converge?

The short answer is: *yes*. Strict diagonal dominance of A is a sufficient condition for the convergence of the Gauss-Seidel method.

To see why this is reasonable, recall that Gauss-Seidel differs from Jacobi only in how aggressively it uses information. Jacobi inverts only the diagonal matrix D, while

Gauss-Seidel inverts the lower triangular matrix $D + L$. From an algorithmic point of view, Gauss-Seidel is therefore a refinement of Jacobi rather than a fundamentally different method. It would be surprising if a matrix structure that guarantees convergence of Jacobi were to fail completely for Gauss-Seidel.

From the convergence framework developed earlier, it suffices to show that the iteration matrix

$$T_{GS} = -(D + L)^{-1}U$$

acts as a contraction in some vector norm. While it is difficult to obtain a simple closed-form expression for $\|T_{GS}\|$ directly—due to the presence of $(D+L)^{-1}$—classical analysis shows that strict diagonal dominance ensures that the influence of the upper triangular part U is not strong enough to overwhelm the stabilizing effect of $D + L$.

More concretely, strict diagonal dominance implies that:

- the matrix $D + L$ is nonsingular;
- the forward substitution process used to compute $(D + L)^{-1}\boldsymbol{r}$ does not amplify errors excessively;
- the combined effect of $(D + L)^{-1}$ and U results in an iteration matrix whose induced infinity norm is strictly less than 1.

Therefore, if A is strictly diagonally dominant by rows, then there exists an induced matrix norm (in particular, the infinity norm) such that

$$\|T_{GS}\| < 1,$$

and the Gauss-Seidel iteration converges for any initial guess.

Remark. Unlike the Jacobi method, where diagonal dominance leads immediately to a transparent row-sum condition, the corresponding argument for Gauss-Seidel is more subtle and typically relies on auxiliary estimates (such as Sassenfeld's condition) rather than a direct entrywise bound on T_{GS}. For this reason, diagonal dominance should be viewed as a reliable *sufficient* condition for convergence, rather than an exact characterization.

In summary, strict diagonal dominance of A guarantees convergence of both Jacobi and Gauss-Seidel. The difference lies not in the result, but in the complexity of the argument required to establish it. This observation motivates the search for sharper and more intrinsic convergence criteria, which we will later express in terms of the spectral radius of the iteration matrix.

13.5 Outlook: A General Convergence Framework

For future iterative algorithms, the moment we can identify the **Iterative Linear Operator**, the moment we can begin analyzing the conditions on the input matrix A so that the iteration converges. Obtaining sufficient conditions for convergence is not hard, but proving "if and only if" conditions require deeper thoughts, and sometimes, it is unattainable. We will see more of convergence analysis once we finish studying two more iterative methods in Lecture 15.

14 Lecture XIV: Successive Over-relaxation and Richardson Residual Form

Building on the geometric intuition from the Gauss-Seidel method, where we visualize each update of our approximation as vectors in $\mathbb{R}^n$, we see that we want to "correct" the previous iterate by moving in the direction towards the true solution. However, a properly guided direction is not good enough. The magnitude of this update, properly scaled, can lead us to the true solution more quickly. Based on this geometric observation and its induced goal, we build a revised Gauss-Seidel method, called **Successive Over-relaxtion** (SOR).

14.1 Building SOR as a Variant of Gauss-Seidel

Under Gauss-Seidel iteration, in matrix formulation,

$$\text{(Gauss-Seidel)} \quad \boldsymbol{x}^{(k+1)} = -(D+L)^{-1} U\boldsymbol{x}^{(k)} + (D+L)^{-1}\boldsymbol{b},$$

we multiply both sides by $D+L$ on the left, and obtain

$$(D+L)\,\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - U\boldsymbol{x}^{(k)} \implies D\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - L\boldsymbol{x}^{(k+1)} - U\boldsymbol{x}^{(k)}.$$

Multiplying D^{-1} on the left of both sides, we have

$$\boldsymbol{x}^{(k+1)} = D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{(k+1)} - U\boldsymbol{x}^{(k)}\right).$$

In practice, we want to control the jump size in the direction of $\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$. We now subtract $\boldsymbol{x}^{(k)}$ from both sides to obtain

$$\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right)_{GS} := \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} = D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{(k+1)} - D\boldsymbol{x}^{(k)} - U\boldsymbol{x}^{(k)}\right). \quad (14.1)$$

One can think of the RHS as the definition of the **Gauss-Seidel Correction**, in the sense that $\boldsymbol{x}^{(k)}$ goes to $\boldsymbol{x}^{(k+1)}$ in this direction. We label this $\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right)_{GS}$ (the Jacobi method will have its own "correction" since the iterative linear operator is different than Gauss-Seidel, but the principle is similar).

Now, from the example we just saw, going from $\boldsymbol{x}^{(k)}$ to $\boldsymbol{x}^{(k+1)}$ using the exact vector $\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right)_{GS}$ may not necessarily yield the "quickest" approach to the true solution. We may need to propel from $\boldsymbol{x}^{(k)}$ a little harder in the direction of $\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$, namely,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \omega\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right)_{GS}, \quad \omega > 1. \quad (14.2)$$

that is, the true landing spot should be in the same direction of the GS Correction but we step on the gas harder (as $\omega > 1$). At this stage, one can implement SOR by

first computing a new iterate from Gauss-Seidel, then the Gauss-Seidel correction via Equation (14.1), and then compute the true update by scaling the Gauss-Seidel with a factor of ω via Equation (14.2). One may refer to SOR as a "scaled Gauss-Seidel".

Altogether, in component form,

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega D^{-1}\left(b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}\right) \\ \implies x^{(k+1)} &= (1-\omega)\, x^{(k)} + \omega D^{-1}\left(b - Lx^{(k+1)} - Ux^{(k)}\right) \end{aligned}$$

In this form, the update $x^{(k+1)}$ can be seen as a weighted average of the previous value $x^{(k)}$ and the vector $D^{-1}\left(b - Lx^{(k+1)} - Ux^{(k)}\right)$ (the update vector). In other words, the updated vector is a prudent deliberation between the previous value and an update vector, where the relaxation parameter plays the role of weights.

14.2 SOR in Matrix Formulation

Using the definition of GS Correction, we then have the update

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \omega D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{(k+1)} - D\boldsymbol{x}^{(k)} - U\boldsymbol{x}^{(k)}\right), \quad \omega > 1.$$

Now, rearranging things a little by dividing both sides by ω and then multiplying on the left by D, we have

$$\begin{aligned} \frac{1}{\omega} D\boldsymbol{x}^{(k+1)} &= \frac{1}{\omega} D\boldsymbol{x}^{(k)} + \boldsymbol{b} - L\boldsymbol{x}^{(k+1)} - D\boldsymbol{x}^{(k)} - U\boldsymbol{x}^{(k)} \\ \implies \left(\frac{1}{\omega} D + L\right)\boldsymbol{x}^{(k+1)} &= \left(\left(\frac{1}{\omega} - 1\right) D - U\right)\boldsymbol{x}^{(k)} + \boldsymbol{b} \\ \overset{\text{multiply by } \omega}{\implies} (D + \omega L)\,\boldsymbol{x}^{(k+1)} &= ((1-\omega)\, D - \omega U)\,\boldsymbol{x}^{(k)} + \omega \boldsymbol{b} \\ \overset{\text{invert } (D+\omega L)}{\implies} \boldsymbol{x}^{(k+1)} &= \boxed{(D + \omega L)^{-1}\left((1-\omega)\, D - \omega U\right)}\boldsymbol{x}^{(k)} + (D + \omega L)^{-1}\,\omega \boldsymbol{b}. \end{aligned}$$

This method is called **Successive over-Relaxation** (**SOR**), where $1 < \omega < 2$, and we note that if $\omega = 1$, we retrieve Gauss-Seidel (check it!).

In the fixed-point form of an iterative method, $\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}$, we identify that the iterative operator/matrix of the SOR method (boxed above) is

$$T_{SOR} = (D + \omega L)^{-1}\left((1-\omega)\, D - \omega U\right)$$

and the constant vector is

$$\boldsymbol{c} = (D + \omega L)^{-1}\,\omega \boldsymbol{b}.$$

14.3 Richardson Residual Form

Up to this point, we have introduced three iterative methods for solving

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

Jacobi, Gauss-Seidel, and Successive Over-Relaxation. Although these methods were derived from different viewpoints – component-wise updates, immediate reuse of new information, and controlled step sizes – they all share two important features:

- they are *stationary* iterations;
- they can all be written in the fixed-point form

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}.$$

While the fixed-point form is convenient for convergence analysis, it obscures a key geometric idea: each update attempts to *correct* the current approximation by responding to how badly it violates the equation $A\boldsymbol{x} = \boldsymbol{b}$. Making this idea explicit leads to a more revealing formulation, known as the **Richardson residual form**.

14.3.1 Residual and correction

Given an approximation $\boldsymbol{x}^{(k)}$, define the **residual**

$$\boldsymbol{r}^{(k)} := \boldsymbol{b} - A\boldsymbol{x}^{(k)}.$$

The residual measures how far $\boldsymbol{x}^{(k)}$ is from satisfying the linear system. A natural iterative strategy is therefore:

use the residual to construct a correction, then update the iterate.

A generic correction step can be written as

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}^{(k)},$$

where $\Delta\boldsymbol{x}^{(k)}$ is constructed from $\boldsymbol{r}^{(k)}$. If we choose a matrix B that approximates A but is easier to invert or solve with, a simple and natural choice is

$$\Delta\boldsymbol{x}^{(k)} = B^{-1}\boldsymbol{r}^{(k)}.$$

This leads to the iteration

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + B^{-1}\big(\boldsymbol{b} - A\boldsymbol{x}^{(k)}\big), \tag{14.3}$$

which is called the **preconditioned Richardson method**.

14.3.2 Why this viewpoint is useful

The Richardson residual form has several advantages:

- it makes the *direction of the update* explicit (through the residual);
- it separates the roles of the matrix A and its approximation B;
- it provides a unifying framework in which Jacobi, Gauss-Seidel, and SOR appear as special cases;
- it connects naturally to modern viewpoints such as preconditioning and gradient-based methods.

In the simplest case, choosing $B = \frac{1}{\tau} I$ yields the basic Richardson iteration

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \tau\big(\boldsymbol{b} - A\boldsymbol{x}^{(k)}\big),$$

where $\tau > 0$ is a step size (often called a *learning rate* in other contexts).

14.3.3 Connection to the fixed-point form

The residual formulation is entirely equivalent to the fixed-point form studied earlier. Expanding Equation (14.3),

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + B^{-1}\boldsymbol{b} - B^{-1}A\boldsymbol{x}^{(k)} = \big(I - B^{-1}A\big)\boldsymbol{x}^{(k)} + B^{-1}\boldsymbol{b}.$$

Thus, the iteration matrix and constant vector are

$$T = I - B^{-1}A, \qquad \boldsymbol{c} = B^{-1}\boldsymbol{b}.$$

The fixed point of this iteration satisfies

$$\boldsymbol{x}^* = T\boldsymbol{x}^* + \boldsymbol{c} \quad \Longleftrightarrow \quad A\boldsymbol{x}^* = \boldsymbol{b},$$

as desired.

14.3.4 Jacobi, Gauss-Seidel, and SOR under the same umbrella

Within the Richardson framework, all three methods correspond simply to different choices of B:

- **Jacobi:** $B = D$,
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + D^{-1}(\boldsymbol{b} - A\boldsymbol{x}^{(k)}).$$

- **Gauss-Seidel:** $B = D + L$,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + (D + L)^{-1}(\boldsymbol{b} - A\boldsymbol{x}^{(k)}).$$

- **SOR:** $B = \frac{1}{\omega}D + L$ (equivalently written after scaling),

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \omega(D + \omega L)^{-1}(\boldsymbol{b} - A\boldsymbol{x}^{(k)}).$$

Seen this way, Jacobi, Gauss-Seidel, and SOR are not three unrelated algorithms, but instances of a single design principle: approximate A by a simpler matrix B, convert the residual into a correction, and iterate. In your homework, you are asked to verify the preconditioner B for each method.

14.3.5 Looking ahead

Both the fixed-point form and the Richardson residual form lead to the same fundamental convergence question: when does the iteration matrix T drive errors to zero? In the next lecture, we will move beyond sufficient norm-based conditions and study convergence through the spectral properties of T, culminating in the punchline theorem:

$$T^k \to 0 \quad \Longleftrightarrow \quad \rho(T) < 1.$$

What the hell is $\rho(T)$? Stay tuned.

15 Lecture XV: Convergence Proofs of Iterative Algorithms for Linear Systems

Let's focus on the scalar-valued sequence $a^{(n)}$ that satisfies the recurrence relation

$$a^{(k+1)} = \alpha a^{(k)}.$$

If $|\alpha| > 1$, the sequence blows up in magnitude. On the other hand, if $|\alpha| \leqslant 1$, then the limit exists; furthermore, if $|\alpha| < 1$, the limit is zero.

Building on the intuition from this example in 1D, now let's study the convergence of any iteration methods.

15.1 Why matrix norms are not the full story

In previous lectures, we studied iterative methods for linear systems written in the fixed-point form

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c},$$

and we established the following sufficient condition for convergence:

$$\|T\| < 1 \quad \Longrightarrow \quad T^k \to 0.$$

This criterion is extremely useful in practice. It allows us to guarantee convergence of Jacobi and Gauss-Seidel iterations under conditions such as strict diagonal dominance. However, it is also clear that this condition is not sharp. In numerical experiments, one often observes convergence even when $\|T\| \geqslant 1$ under a particular choice of norm.

This observation motivates a deeper question: is there an intrinsic quantity associated with the matrix T, independent of any chosen norm, that truly governs convergence?

To answer this, we return to the object that actually propagates errors.

15.2 Error propagation and eigenmodes

Let $\boldsymbol{x}^*$ denote the true solution of the linear system. By definition, it satisfies

$$\boldsymbol{x}^* = T\boldsymbol{x}^* + \boldsymbol{c}. \tag{15.1}$$

Define the error vector

$$\boldsymbol{e}^{(k)} := \boldsymbol{x}^{(k)} - \boldsymbol{x}^*.$$

Substituting $\boldsymbol{x}^{(k)}$ in terms of the error and true solution into the fixed-point form in Equation (15.1) yields

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= T\boldsymbol{x}^{(k)} + \boldsymbol{c} \\ \implies \boldsymbol{x}^* + e^{(k+1)} &= T\left(\boldsymbol{x}^* + e^{(k)}\right) + \boldsymbol{c} \\ \implies e^{(k+1)} &= Te^{(k)} + \cancelto{0}{(T\boldsymbol{x}^* + \boldsymbol{c} - \boldsymbol{x}^*)} \end{aligned}$$

and hence

$$\boldsymbol{e}^{(k)} = T^k \boldsymbol{e}^{(0)}, \tag{15.2}$$

by repeated multiplication of T. This shows how error propagates.

At this point, we recall several facts from linear algebra that will guide our intuition:

- the definition of eigenvalues and eigenvectors,
- diagonalization of matrices,
- expressing vectors in a basis of eigenvectors.

For the sake of intuition, we temporarily assume that T is **diagonalizable**. That is, there exists an invertible matrix V such that

$$T = V\Lambda V^{-1}, \qquad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n),$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of T.

15.3 Decomposition of the error

Any initial error vector $\boldsymbol{e}^{(0)}$ can be expanded in the eigenvector basis,

$$\boldsymbol{e}^{(0)} = \sum_{j=1}^{n} \alpha_j \boldsymbol{v}_j,$$

where $\boldsymbol{v}_j$ is an eigenvector corresponding to λ_j. Thus, Equation (15.2) yields

$$\boldsymbol{e}^{(k)} = T^k \boldsymbol{e}^{(0)} = \sum_{j=1}^{n} \alpha_j \lambda_j^k \boldsymbol{v}_j,$$

by appealing to the fact that $T^k \boldsymbol{v}_j = \lambda^k \boldsymbol{v}_j$ (why?).

This expression reveals that each eigenvalue λ_j represents an independent growth or decay mode of the error. Some components may decay rapidly, others slowly, and some may not decay at all.

15.4 Factoring out the dominant eigenvalue

Order the eigenvalues by magnitude:

$$|\lambda_1| \geqslant |\lambda_2| \geqslant \cdots \geqslant |\lambda_n|.$$

Factoring out λ_1^k, we obtain

$$\boldsymbol{e}^{(k)} = \lambda_1^k \left(\alpha_1 \boldsymbol{v}_1 + \sum_{j=2}^{n} \alpha_j \left(\frac{\lambda_j}{\lambda_1} \right)^k \boldsymbol{v}_j \right).$$

If $|\lambda_1| > |\lambda_2|$, then

$$\left| \frac{\lambda_j}{\lambda_1} \right| < 1 \quad \text{for all } j \geqslant 2,$$

and therefore

$$\left(\frac{\lambda_j}{\lambda_1} \right)^k \to 0 \quad \text{as } k \to \infty.$$

Consequently,

$$\frac{\boldsymbol{e}^{(k)}}{\lambda_1^k} \longrightarrow \alpha_1 \boldsymbol{v}_1.$$

This shows that, asymptotically, the behavior of the error is governed entirely by the eigenvalue of largest magnitude.

15.5 The spectral radius

The preceding analysis strongly suggests that convergence of the iteration depends on the largest eigenvalue magnitude of T. This motivates the following definition.

Definition 15.1 (Spectral Radius)**.** *The spectral radius of a matrix T is defined as*

$$\rho(T) := \max\{\, |\lambda| : \lambda \textit{ is an eigenvalue of } T \,\}.$$

Geometrically, $\rho(T)$ is the radius of the smallest circle centered at the origin in the complex plane that contains all eigenvalues of T.

15.6 Why the spectral radius matters

From the error representation

$$\boldsymbol{e}^{(k)} = \sum_{j=1}^{n} \alpha_j \lambda_j^k \boldsymbol{v}_j,$$

we see that convergence of the iterative method for every initial guess requires

$$|\lambda_j| < 1 \quad \text{for all eigenvalues } \lambda_j.$$

Equivalently, we require

$$\rho(T) < 1.$$

This observation explains why matrix norms provide only sufficient conditions for convergence, while the spectral radius captures the true asymptotic behavior of the iteration.

This result leads itself to the single most important theorem for iterative methods: repeated multiplications of a matrix T on a vector $\boldsymbol{x}$ yields the $\mathbf{0}$ vector **if and only if** the spectral radius is strictly less than 1.

Theorem 15.1 (The Punchline Theorem for Iterative Methods for Linear Systems)**.** *Given $T \in \mathbb{R}^{n\times n}$, then $\rho(T) < 1$ iff $\lim_{n\to\infty} T^n \boldsymbol{x} = \mathbf{0}$ for every $\boldsymbol{x} \in \mathbb{R}^n$.*

You may recall the scalar version of this theorem to help you understand

$$|\alpha| < 1 \iff \lim_{k\to\infty} \alpha^k c = 0$$

for any constant c.

15.7 Proof of the Punchline Theorem

Assuming that T is diagonalizable, we prove both sufficiency and necessity.

15.7.1 Sufficiency: $\rho(T) < 1 \implies T^k \to 0$

Assume that the iteration matrix T is diagonalizable and that its spectral radius satisfies

$$\rho(T) < 1.$$

Then there exists an invertible matrix V such that

$$T = V\Lambda V^{-1}, \qquad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n),$$

with $|\lambda_j| < 1$ for all $j = 1, \ldots, n$.

Raising T to the kth power gives

$$T^k = V\Lambda^k V^{-1}, \qquad \Lambda^k = \operatorname{diag}(\lambda_1^k, \ldots, \lambda_n^k).$$

Since $|\lambda_j| < 1$, we have $\lambda_j^k \to 0$ as $k \to \infty$ for each j. Hence,

$$\Lambda^k \to 0,$$

and therefore

$$T^k = V\Lambda^k V^{-1} \to 0.$$

It follows that for any initial error $\boldsymbol{e}^{(0)}$,

$$\boldsymbol{e}^{(k)} = T^k \boldsymbol{e}^{(0)} \to \mathbf{0},$$

and thus the iterative method converges for every initial guess.

This proves that $\rho(T) < 1$ is a sufficient condition for convergence.

15.7.2 Necessity: $T^k \to 0 \implies \rho(T) < 1$

We now prove the converse. Assume that

$$T^k \to 0 \quad \text{as } k \to \infty.$$

Suppose, for contradiction, that $\rho(T) \geqslant 1$. Then there exists an eigenvalue λ of T such that $|\lambda| \geqslant 1$, with corresponding (nonzero) eigenvector $\boldsymbol{v}$ satisfying

$$T\boldsymbol{v} = \lambda \boldsymbol{v}.$$

Applying T^k to $\boldsymbol{v}$ yields

$$T^k \boldsymbol{v} = \lambda^k \boldsymbol{v}.$$

If $|\lambda| > 1$, then $\|\lambda^k \boldsymbol{v}\| \to \infty$; if $|\lambda| = 1$, then $\|\lambda^k \boldsymbol{v}\| = \|\boldsymbol{v}\|$ for all k. In either case,

$$T^k \boldsymbol{v} \not\to \mathbf{0}.$$

This contradicts the assumption that $T^k \to 0$ as an operator, since convergence to zero must hold when applied to every vector. Therefore, no eigenvalue of T can satisfy $|\lambda| \geqslant 1$, and we must have

$$\rho(T) < 1.$$

This shows that $\rho(T) < 1$ is also a necessary condition for convergence.

15.8 The relationship between $\rho(T)$ and $\|T\|$

We now clarify the relationship between the spectral radius of a matrix and its matrix norm. Recall that for any induced matrix norm $\|\cdot\|$, the following inequality always holds:

$$\rho(T) \leqslant \|T\|.$$

This inequality explains why matrix norms naturally appear in convergence analysis: if $\|T\| < 1$, then automatically $\rho(T) < 1$, and hence the iteration converges.

However, the converse is not true in general. That is,

$$\rho(T) < 1 \quad \not\Longrightarrow \quad \|T\| < 1.$$

Matrix norms therefore provide *sufficient* but not *necessary* conditions for convergence.

An illustrative example

Consider the matrix

$$T = \begin{bmatrix} 0 & 1 \\ \frac{1}{2} & 0 \end{bmatrix}.$$

The eigenvalues of T are $\lambda = \pm\frac{1}{\sqrt{2}}$, so

$$\rho(T) = \frac{1}{\sqrt{2}} < 1.$$

Therefore, $T^k \to 0$, and any fixed-point iteration with this iteration matrix converges.

On the other hand, consider the matrix infinity norm:

$$\|T\|_\infty = \max\left\{|0| + |1|,\ \left|\tfrac{1}{2}\right| + |0|\right\} = 1.$$

In fact, under other induced norms, one may even find $\|T\| > 1$. **Despite this, the iteration still converges, because the true governing quantity is the spectral radius, not the norm.**

Interpretation

This example highlights the roles played by $\rho(T)$ and $\|T\|$:

- Matrix norms are easy to compute and lead to clean sufficient conditions.
- The spectral radius captures the true asymptotic behavior of T^k.
- Convergence can occur even when $\|T\| \geqslant 1$, provided $\rho(T) < 1$.

In summary, matrix norms are useful tools for analysis, but the spectral radius is the fundamental quantity that determines convergence of linear fixed-point iterations.

15.9 Rate of Convergence

The rate of convergence in fact relates to **spectral radius** of the iterative linear operator as well. So, it boils down to computing their spectral radius, e.g. that of $T_J = -D^{-1}(L+U)$ for Jacobi, and $T_{GS} = -(D+L)^{-1}U$ for Gauss-Seidel.

A theorem relating the spectral radius and the matrix norms are given by the following Theorem.

Theorem 15.2. *If A is an $n \times n$ matrix, then*

1. $\|A\|_2 = \left[\rho\left(A^{\mathrm{T}}A\right)\right]^{1/2}$,
2. $\rho(A) \leqslant \|A\|$, *for any induced norm* $\|\cdot\|$.

The result we will use is item 1 because finally we have connected the elusive matrix 2-norm to the spectral radius. More examples are shown in the homework because the computation is a straightforward process of finding eigenvalues of the Gramian matrix $A^{\mathrm{T}}A$.

The following theorem characterizes the rate of convergence based on the matrix norm of the Iterative Linear Operator. Due to the theorem above, the spectral radius may then act as a proxy for the matrix norm, and thus suggest the rate of convergence.

Theorem 15.3. *If $\|T\| < 1$ for any induced norm $\|\cdot\|$ and $\boldsymbol{c}$ is a given vector, then $\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}$ converges, for any $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$, to $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{x} = T\boldsymbol{x} + \boldsymbol{c}$ (also known as a fixed point of the map T). Furthermore, we have the following error bounds*

1. $\left\|\boldsymbol{x} - \boldsymbol{x}^{(k)}\right\| \leqslant \|T\|^k \left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}\right\|$;
2. $\left\|\boldsymbol{x} - \boldsymbol{x}^{(k)}\right\| \leqslant \frac{\|T\|^k}{1-\|T\|} \left\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\right\|$.

Since we know there is deep connection between the matrix norm $\|T\|$ and the spectral radius $\rho(T)$, we arrive at the following error estimate

$$\left\|\boldsymbol{x} - \boldsymbol{x}^{(k)}\right\| \approx \left[\rho(T)\right]^k \left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}\right\|.$$

15.10 Another View of Convergence via Repeated Application of T

For any iterative algorithms, we identify its the iteration operator/matrix T as

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}$$

where $\boldsymbol{x}^{(0)}$ is an arbitrary initial guess and $\boldsymbol{c}$ a constant vector (that involves b from $Ax = b$ in the context of system of equations).

Let's apply this formula iteratively. Noting that $\boldsymbol{x}^{(k)} = T\boldsymbol{x}^{(k-1)} + \boldsymbol{c}$, we substitute this into the iteration and obtain

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= T\left(T\boldsymbol{x}^{(k-1)} + \boldsymbol{c}\right) + \boldsymbol{c} \\
&= T^2\boldsymbol{x}^{(k-1)} + (T + I)\,\boldsymbol{c} \\
&\cdot \\
&\cdot \\
&\cdot \\
&= T^k\boldsymbol{x}^{(0)} + \left(T^{k-1} + T^{k-2} + \cdots + T + I\right)\boldsymbol{c}
\end{aligned}$$

Note by the previous theorem, if $\rho(T) < 1$, we must have $T^k\boldsymbol{x}^{(0)} \to \boldsymbol{0}$ as $k \to \infty$. Meanwhile, the sum

$$\sum_{i=0}^{k-1} T^i \to \sum_{i=1}^{\infty} T^i = (I - T)^{-1}$$

by the geometric series formula (for matrices – you can find the scalar version in any Calculus textbook or Wikipedia). Therefore,

$$\lim_{k\to\infty} \boldsymbol{x}^{(k+1)} = \lim_{k\to\infty} T^k\boldsymbol{x}^{(0)} + \left(\sum_{i=1}^{\infty} T^i\right)\boldsymbol{c} = \boldsymbol{0} + (I - T)^{-1}\,\boldsymbol{c},$$

proving that $\boldsymbol{x}^{(k+1)}$ has an explicit limit (depending on A and b, which are given information)

$$\boldsymbol{x} = (I - T)^{-1}\,\boldsymbol{c} \implies (I - T)\,\boldsymbol{x} = \boldsymbol{c} \implies \boldsymbol{x} = T\boldsymbol{x} + \boldsymbol{c}.$$

This procedure proves that for an iterative scheme to converge, a sufficient condition is $\rho(T) < 1$. In fact, one can show that this condition is also necessary.

16 Lecture XVI: Power Method for Eigenvalue

16.1 Eigenvalues and Eigenvectors: A Review

Matrix operations on vectors have deep geometric meanings. For each matrix, there may be certain vectors that don't change direction at all, but only gets stretched or compressed. In other words,

$$A\boldsymbol{x} = \lambda \boldsymbol{x}$$

where $\lambda \in \mathbb{R}$ is a scaling factor that describes the extent of stretching or compression. We call λ the eigenvalue of A corresponding to the eigenvector $\boldsymbol{x}$.

Rearranging the equation, we have

$$A\boldsymbol{x} - \lambda \boldsymbol{x} = 0 \implies A\boldsymbol{x} - \lambda I \boldsymbol{x} = 0$$

where I is the identity matrix in $\mathbb{R}^n$. Then, we can factor out $\boldsymbol{x}$ such that

$$(A - \lambda I)\, \boldsymbol{x} = 0,$$

which is just another system of equations $\tilde{A}\boldsymbol{x} = 0$. This equation has a unique solution if and only if $\det \tilde{A} = \det (A - \lambda I) = 0$, i.e., the determinant of $(A - \lambda I)$ is zero. Note that $A - \lambda I$ only differs from A on the diagonal entries (since the identity only affects the diagonal entries).

The determinant condition yields a root-finding problem of a polynomial in λ. For each eigenvalue λ, we associate its eigenvector $\boldsymbol{v}$ that satisfies

$$A\boldsymbol{v} = \lambda \boldsymbol{v}.$$

From the last few lectures on iterative methods for linear systems, we saw that the defining theorem of the section is that the spectral radius controls the convergence of iterative methods. It is thus of paramount importance to develop mathematical and computational procedures to obtain the spectral radius (and other eigenvalues). With this goal in mind, we construct the most fundamental algorithm of eigenvalue determination.

16.2 The Power Method for Spectral Radius

Given an $n \times n$ square matrix A, assume that it has ordered eigenvalues

$$|\lambda_1| > |\lambda_2| \geqslant |\lambda_3| \geqslant \cdots \geqslant |\lambda_n| \geqslant 0$$

with associated eigenvectors $\boldsymbol{v}_1, \ldots \boldsymbol{v}_n$ that are **normalized** ($\|\boldsymbol{v}_i\|_\infty = 1$ – to normalize a vector in the l^∞ sense, we divide each component by the maximal entry in absolute

value, namely, $\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_\infty}$). The $\boldsymbol{v}_i$'s are not necessarily mutually linearly independent since we allow the subordinate eigenvalues to be possibly equal. However, as the dominant eigenvalue (the value itself) is unique, for any $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$, we can find coefficients β_i's such that

$$\boldsymbol{x}^{(0)} = \sum_{i=1}^{n} \beta_i \boldsymbol{v}_i.$$

where $\beta_1 \neq 0$.

Now, let's multiply both sides by A on the left, and use the eigenvector-eigenvalue relationship,

$$A\boldsymbol{x}^{(0)} = A\sum_{i=1}^{n} \beta_i \boldsymbol{v}_i = \sum_{i=1}^{n} \beta_i A\boldsymbol{v}_i = \sum_{i=1}^{n} \beta_i \lambda_i \boldsymbol{v}_i.$$

Let's apply A again on both sides and obtain

$$A^2\boldsymbol{x}^{(0)} = \sum_{i=1}^{n} \beta_i \lambda_i A\boldsymbol{v}_i = \sum_{i=1}^{n} \beta_i \lambda_i^2 \boldsymbol{v}_i.$$

After doing this k times, we have

$$A^k\boldsymbol{x}^{(0)} = \sum_{i=1}^{n} \beta_i \lambda_i^k \boldsymbol{v}_i.$$

Knowing that λ_1 has the largest magnitude, let's factor it out as in

$$A^k\boldsymbol{x}^{(0)} = \lambda_1^k \sum_{i=1}^{n} \beta_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \boldsymbol{v}_i = \lambda_1^k \beta_1 \boldsymbol{v}_1 + \lambda_1^k \left(\beta_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k \boldsymbol{v}_2 + \cdots + \beta_n \left(\frac{\lambda_n}{\lambda_1}\right)^k \boldsymbol{v}_n\right).$$

As k grows, the contribution from $\left(\frac{\lambda_i}{\lambda_1}\right)^k$ diminishes for $i = 2, 3, \ldots, n$, and the RHS becomes dominated by the leading term ($i = 1$), that is, by defining $\boldsymbol{x}^{(k)} = A^k\boldsymbol{x}^{(0)}$, we have

$$\lim_{k\to\infty} \boldsymbol{x}^{(k)} = \lim_{k\to\infty} A^k\boldsymbol{x}^{(0)} = \lim_{k\to\infty} \lambda_1^k \beta_1 \boldsymbol{v}_1.$$

This is saying that when k is large enough, $A^k\boldsymbol{x}^{(0)}$ is almost parallel to $\boldsymbol{v}_1$.

We observe that the limit on the RHS only converges if $|\lambda_1| < 1$ and $\beta_1 \neq 0$. Meanwhile, the limit on the LHS may incur overflow or underflow since the entries of A can easily blow up from repeated matrix multiplications. To avoid this, we rescale the product $A^k\boldsymbol{x}^{(0)}$ by its infinity norm $\left\|A^{k-1}\boldsymbol{x}^{(0)}\right\|_\infty$, so that we always iterating the multiplication to a unit vector in l^∞.

With the rescaling, define $\boldsymbol{z}^{(k)} = \frac{A^k \boldsymbol{x}^{(0)}}{\left\| A^{k-1} \boldsymbol{x}^{(0)} \right\|_\infty}$, we see that

$$\lim_{k\to\infty} \boldsymbol{z}^{(k)} = \lim_{k\to\infty} \frac{A^k \boldsymbol{x}^{(0)}}{\left\| A^{k-1} \boldsymbol{x}^{(0)} \right\|_\infty} = \lim_{k\to\infty} \frac{\lambda_1^k \beta_1 \boldsymbol{v}_1}{\left\| \lambda_1^{k-1} \beta_1 \boldsymbol{v}_1 \right\|_\infty} = \lambda_1 \boldsymbol{v}_1,$$

namely, the rescaled iteration leads to the principal eigenvector corresponding to the spectral radius of A.

Furthermore, define $\boldsymbol{y}^{(k)} = \frac{A^k \boldsymbol{x}^{(0)}}{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty}$, we find that

$$\lim_{k\to\infty} \boldsymbol{y}^{(k)} = \lim_{k\to\infty} \frac{A^k \boldsymbol{x}^{(0)}}{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty} = \lim_{k\to\infty} \frac{\lambda_1^k \beta_1 \boldsymbol{v}_1}{\left\| \lambda_1^k \beta_1 \boldsymbol{v}_1 \right\|_\infty} = \boldsymbol{v}_1$$

which shows that the sequence of $\boldsymbol{y}^{(k)}$ (as a result of repeated multiplication of A) converges to the principle eigenvector of A, corresponding to the eigenvalue with maximal magnitude. Know the limits of both $\boldsymbol{x}^{(k)}$ and $\boldsymbol{z}^{(k)}$ determines λ_1 since for k large enough, we must have

$$\lambda_1 \boldsymbol{y}^{(k)} \approx \boldsymbol{z}^{(k)} \implies \lambda_1 \approx \frac{\left\| \boldsymbol{z}^{(k)} \right\|_\infty}{\left\| \boldsymbol{y}^{(k)} \right\|_\infty} = \frac{\frac{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty}{\left\| A^{k-1} \boldsymbol{x}^{(0)} \right\|_\infty}}{\frac{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty}{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty}} = \frac{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty}{\left\| A^{k-1} \boldsymbol{x}^{(0)} \right\|_\infty} = \frac{\left\| \boldsymbol{x}^{(k)} \right\|_\infty}{\left\| \boldsymbol{x}^{(k-1)} \right\|_\infty},$$

that is, the leading eigenvalue is equal to ratio of the infinity norm of the successive iterates. The method of determining λ_1 is called the **Power Method**.

Example 16.1. Let $A = \begin{bmatrix} -2 & -3 \\ 6 & 7 \end{bmatrix}$. We know its eigenvalues $\lambda_1 = 4$ and λ_2, with corresponding eigenvectors $\boldsymbol{v}_1 = (1, -2)^{\mathrm{T}}$ and $\boldsymbol{v}_2 = (1, -1)^{\mathrm{T}}$. Suppose we start with $\boldsymbol{x}^{(0)} = (1, 1)^{\mathrm{T}}$.

$$
\begin{aligned}
\boldsymbol{x}^{(1)} &= A\boldsymbol{x}^{(0)} = \begin{bmatrix} -5 \\ 13 \end{bmatrix}, \\
\boldsymbol{x}^{(2)} &= A\boldsymbol{x}^{(1)} = \begin{bmatrix} -29 \\ 61 \end{bmatrix}, \\
\boldsymbol{x}^{(3)} &= A\boldsymbol{x}^{(2)} = \begin{bmatrix} -125 \\ 253 \end{bmatrix}, \\
\boldsymbol{x}^{(4)} &= A\boldsymbol{x}^{(3)} = \begin{bmatrix} -509 \\ 1021 \end{bmatrix}, \\
\boldsymbol{x}^{(5)} &= A\boldsymbol{x}^{(4)} = \begin{bmatrix} -2045 \\ 4093 \end{bmatrix}, \\
\boldsymbol{x}^{(6)} &= A\boldsymbol{x}^{(5)} = \begin{bmatrix} -8189 \\ 16381 \end{bmatrix}.
\end{aligned}
$$

By the derivation above, we estimate the eigenvalue with maximal magnitude

$$
\begin{aligned}
\lambda_1^{(1)} &= \frac{\left\|\boldsymbol{x}^{(2)}\right\|_\infty}{\left\|\boldsymbol{x}^{(1)}\right\|_\infty} = \frac{61}{13} = 4.6923 \\
\lambda_1^{(2)} &= \frac{\left\|\boldsymbol{x}^{(3)}\right\|_\infty}{\left\|\boldsymbol{x}^{(2)}\right\|_\infty} = \frac{253}{61} = 4.1475 \\
\lambda_1^{(3)} &= \frac{\left\|\boldsymbol{x}^{(4)}\right\|_\infty}{\left\|\boldsymbol{x}^{(3)}\right\|_\infty} = \frac{1021}{253} = 4.03557 \\
\lambda_1^{(4)} &= \frac{\left\|\boldsymbol{x}^{(5)}\right\|_\infty}{\left\|\boldsymbol{x}^{(4)}\right\|_\infty} = \frac{4093}{1021} = 4.00881 \\
\lambda_1^{(5)} &= \frac{\left\|\boldsymbol{x}^{(6)}\right\|_\infty}{\left\|\boldsymbol{x}^{(5)}\right\|_\infty} = \frac{16381}{4093} = 4.00200.
\end{aligned}
$$

and $\boldsymbol{x}^{(6)} = \begin{bmatrix} -8189 \\ 16381 \end{bmatrix}$ which normalizes to $\begin{bmatrix} -0.49908 \\ 1 \end{bmatrix} \approx \boldsymbol{v}_1$.

This examples shows that rescaling is absolutely necessary when we expect the method to converge slowly because the values in the iterates can easily blow up. The derivation made above is useful to give us a sense of how things may converge, but is not useful in practice. Below, we conclude the power method in an algorithm with careful considerations of rescaling.

16.3 Algorithm

In practice, we compute $\boldsymbol{x}^{(k)}$ iteratively. Given an initial guess $\boldsymbol{x}$,

1. Find the first index p such that $|x_p| = \|\boldsymbol{x}\|_\infty$.
   (a) Rescale $\boldsymbol{x} = \boldsymbol{x}/x_p$.
   (b) Begins iteration $\boldsymbol{y} = A\boldsymbol{x}$.
   (c) Set $\mu = y_p$. Find the first index p such that $|y_p| = \|\boldsymbol{y}\|_\infty$.
      i. If $y_p = 0$, then output eigenvector $\boldsymbol{x}$ and an eigenvalue of 0. We should choose a different initial guess.
      ii. Otherwise, set
      $$error = \|\boldsymbol{x} - \boldsymbol{y}/y_p\|_\infty$$
      and $\boldsymbol{x} = \boldsymbol{y}/y_p$ and return to step 3 until $error < tol$.
   (d) Output $(\mu, \boldsymbol{x})$ as the eigenvalue-eigenvector pair.

16.4 Convergence

Using the final result of the derivation, recall that $\boldsymbol{y}^{(k)} = \frac{A^k \boldsymbol{x}^{(0)}}{\left\| A^k \boldsymbol{x}^{(0)} \right\|_\infty}$ and we know it converges to the principal eigenvector $\boldsymbol{v}_1$. Indeed,

$$
\begin{aligned}
\left\| \boldsymbol{y}^{(k)} - \boldsymbol{v}_1 \right\| &= \left\| \frac{A^k \boldsymbol{x}^{(0)}}{\| A^k \boldsymbol{x}^{(0)} \|_\infty} - \boldsymbol{v}_1 \right\| \\
&= \left\| \frac{\sum_{i=1}^n \beta_i \lambda_i^k \boldsymbol{v}_i}{\sum_{i=1}^n \beta_i \lambda_i^k} - \boldsymbol{v}_1 \right\| \\
&= \left\| \frac{\lambda_1^k \sum_{i=1}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \boldsymbol{v}_i}{\lambda_1^k \sum_{i=1}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k} - \boldsymbol{v}_1 \right\| \\
&= \left\| \frac{\sum_{i=1}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \boldsymbol{v}_i}{\sum_{i=1}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k} - \boldsymbol{v}_1 \right\| \\
&= \left\| \frac{\beta_1 \boldsymbol{v}_1 + \sum_{i=2}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \boldsymbol{v}_i}{\beta_1 + \sum_{i=2}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k} - \boldsymbol{v}_1 \right\| \\
&= \left\| \frac{\sum_{i=2}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \boldsymbol{v}_i - \sum_{i=2}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \boldsymbol{v}_1}{\beta_1 + \sum_{i=2}^n \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k} \right\| \\
&\leqslant \left\| \frac{\sum_{i=2}^n \beta_i \left| \frac{\lambda_i}{\lambda_1} \right|^k (\boldsymbol{v}_i - \boldsymbol{v}_1)}{\beta_1} \right\| \\
&\leqslant \left\| \left| \frac{\lambda_2}{\lambda_1} \right|^k \frac{\sum_{i=2}^n \beta_i (\boldsymbol{v}_i - \boldsymbol{v}_1)}{\beta_1} \right\|, \quad \text{since } \lambda_i \leqslant \lambda_2, \quad i = 2, \ldots, n. \\
&\leqslant \left| \frac{\lambda_2}{\lambda_1} \right|^k \left\| \frac{\sum_{i=2}^n \beta_i}{\beta_1} \right\| (\| \boldsymbol{v}_i \| + \| \boldsymbol{v}_1 \|) \\
&= C \left| \frac{\lambda_2}{\lambda_1} \right|^k
\end{aligned}
$$

where we call $C = 2 \left| \frac{\sum_{i=2}^n \beta_i}{\beta_1} \right|$. Thus, the rate of convergence depends on the ratio of $\frac{\lambda_2}{\lambda_1}$ (in fact the rate is $O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right)$). If λ_2 is smaller but very close to λ_1 in magnitude,

then the convergence is expected to be slow.

Part IV
Root-finding Algorithms

In the ending lecture of iterative methods, we touched upon a fundamental iterative method in finding the dominant eigenvalue, i.e., the spectral radius, of a matrix, even though the typical procedure of finding eigenvalues is to find the zeros of a polynomial by definition (the solvability condition for $(A - \lambda I)\, v = 0$). The iterative method was useful, but requires plenty computational power due to the matrix-vector product, at **every** step of the iteration. Can we resort to what's fundamentally associated with the problem, that is, **finding the zeros of a polynomial**? We investigate several seminal methods to achieve this goal in this part of the course.

In scientific inquiries, we often encounter the need to find the zeros of a function, i.e., $\{x \in \mathbb{R}^n \mid f(\boldsymbol{x}) = 0\}$ where $f : \mathbb{R}^n \to \mathbb{R}$. This type of problem is called **Root-Finding**. A typical example is from optimization. Consider a differentiable function $y = f(x)$ in 1D. Its local optima occur at critical points, which satisfy $f'(x) = 0$. In higher dimensions, we solve simultaneous equations as we set $\nabla f = \mathbf{0}$, the zero vector. The problem becomes increasingly hard not only as dimension increases, but also as the nonlinearity of f gets more complicated.

17 Lecture XVII: Fixed-Point Iteration

17.1 Fixed Points

The notion of fixed points is central to numerical analysis. It appears quite often in convergence proofs of iterative algorithms, such as the Jacobi method, the Gauss-Seidel method, and Successive over-Relaxation (SOR), and later, Newton's method.

Definition 17.1. *The number p is a fixed point for a given function g if $g(p) = p$.*

Example 17.1. Consider an iterative scheme with an iterative linear operator T, e.g., Jacobi method has $T_J = D^{-1}(L + U)$. The iterative algorithm satisfies

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}.$$

Consider a function

$$g\left(\boldsymbol{x}^{(k)}\right) = T\boldsymbol{x}^{(k)} + \boldsymbol{c}.$$

Clearly, if $\boldsymbol{x}^{(k+1)}$ converges to $\boldsymbol{x}$, we must achieve

$$\boldsymbol{x} = T\boldsymbol{x} + c \iff \boldsymbol{x} = g(\boldsymbol{x}) = T\boldsymbol{x} + \boldsymbol{c},$$

that is, the limit is a fixed point of the function g.

In this section, we consider the problem of finding solutions to fixed-point problems and the connection between the fixed-point problems and root-finding problems we wish to solve. Root-finding problems and fixed-point problems are equivalent classes in the following sense:

- Given a root-finding problem $f(x) = 0$, we can define a function g with a fixed point at x **in a number of ways**, for example, as

$$g(x) = x - f(x), \text{or } g(x) = x + 3f(x).$$

- Conversely, if the function g has a fixed point at p, then the function defined by

$$f(x) = x - g(x)$$

  has a zero at p.

Remark 17.1 (Important Remark)**.** *One must note that for a single root-finding problem $f(p) = 0$, there are **more than one way** to convert to a fixed-point problem, i.e. many many choices of $g(x)$. In fact, the choice affects the rate of convergence (to be studied later).*

The reason why we put root-finding problems in the form of fixed-point form is because they are then easier to analyze (such as the convergence analysis of iterative methods for linear systems). Problems easier to analyze tend to give rise to better algorithms. Let's first see a few examples pertaining to just the definition of fixed points.

Example 17.2. Determine any fixed points of the function $g(x) = x^2 - 2$.

Solution. A fixed point p for g satisfies

$$p = g(p) = p^2 - 2$$

which implies that $p^2 - p - 2 = 0 \implies p = 2, -1$.

Remark 17.2. *The fixed point(s) of a function $g(x)$ occur at the intersection between itself and the line $y = x$.*

Example 17.3 (Converting a Root-finding Problem to a Fixed-point Problem)**.** We are tasked to find the root of a polynomial $f(x) = x^3 + 2x - 1$ on $[0, 1]$. Bisection could work but it may be too slow. Let's learn how to convert it to a fixed point problem.

1. So, the task is to "solve" $x^3 + 2x - 1 = 0$. Let's choose

$$g_1(x) = x^3 + x - 1.$$

The fixed point of g satisfies

$$x = g_1(x) = x^3 + 3x - 1 \iff x^3 + 2x - 1 = 0.$$

This suggests that the root of f is a fixed point of g_1.

2. We can also try to isolate a single x from f since the fixed point formulation contains $x = g(x)$. In our case here,

$$x^3 + 2x - 1 = 0 \iff x = \frac{1 - x^3}{2}.$$

So, we may choose $g_2(x) = \frac{1-x^3}{2}$.

3. This is not the only choice. Consider

$$g_3(x) = x\left(x^3 + 2x\right)$$

The fixed point of g_3 satisfies

$$x = g_3(x) = x\left(x^3 + 2x\right) \iff x^3 + 2x = 1 \iff x^3 + 2x - 1 = 0$$

should $x \neq 0$.

These are all valid choices for g, but do they all guarantee that the fixed point of g lies in the intended interval of search? Not always. Next, we provide a sufficient condition for the existence and uniqueness of a fixed point.

17.2 Existence and Uniqueness of Fixed Points

Is there a special class of functions or conditions such that a fixed point always exists? Let's draw any function that maps between the same set $[a, b]$. Putting the line $y = x$ through it seems to always intersect the function. The proof is instructive, and should be understood (without worrying about its use as it is mathematically beautiful).

Theorem 17.1. *(Existence and Uniqueness of Fixed Points)*

1. *(existence) If $g \in C[a, b]$ and $g(x) \in [a, b]$ for all $x \in [a, b]$, i.e.*

$$g : [a, b] \to [a, b],$$

then g has at least one fixed point in $[a, b]$.

2. *(uniqueness) If, in addition, $g'(x)$ exists on (a,b) and a positive constant $k<1$ exists with*

$$|g'(x)| \leqslant k, \quad \textit{for all } x \in (a,b),$$

then there is exactly one fixed point in $[a,b]$.

Proof. We begin with the first statement.

1. If $g(a) = a$ or $g(b) = b$, then g has a fixed point at an endpoint, and we are done. If not, then $g(a) > a$ and $g(b) < b$ (because if $g(a) < a$ or $g(b) > b$, then $g : [a,b] \to [a_-, b_+]$ which is not our assumption).
   Consider function $h(x) = g(x) - x$. It is continuous on $[a,b]$, with

$$\begin{aligned} h(a) &= g(a) - a > 0, \\ h(b) &= g(b) - b < 0. \end{aligned}$$

   Intermediate value theorem then informs us that there exists (at least one) $p \in (a,b)$ such that $h(p) = 0$. This p is a fixed point of g because

$$0 = h(p) = g(p) - p \implies g(p) = p.$$

2. We have just proved that there is at least one fixed point of $g(x)$. Now with further regularity (smoothness) on g such that $|g'(x)| \leqslant k < 1$, we proceed to prove that the fixed point, if exists, is unique.
   We suppose that there are **two fixed points** p and q that both lie in $[a,b]$ for g. If $p \neq q$, say $p > q$, then the Mean Value Theorem implies that there exists $\xi \in (q,p) \subset [a,b]$ such that

$$\frac{g(p) - g(q)}{p - q} = g'(\xi).$$

   Thus, using the fact that $g(p) = p$ and $g(q) = q$, we find

$$|p - q| = |g(p) - g(q)| = |g'(\xi)|\,|p - q| \leqslant k\,|p - q| < |p - q|,$$

   which is impossible since k is **strictly** less than 1. This contradiction must come from the only assumption that $p \neq q$. Hence, $p = q$, and the fixed point in $[a,b]$ is unique.

□

Remark 17.3. *The condition (for existence of a fixed point) that g maps an interval to itself is crucial – that is, g takes in **all** values on $[a,b]$, and achieves $g(x) \in [a,b]$. Visually speaking, this requirement along with continuity forces the graph of g to connect the vertical lines $x = a$ to $x = b$, which, in turn, forces g to cross the line $y = x$.*

Remark 17.4. *Note that this proof is very similar to the use of Rolle's Theorem when proving that there exists exactly one root for certain function $f(x)$ on some interval $[a, b]$. In fact, these two procedures are equivalent.*

Example 17.4. Show that $g(x) = \frac{x^2-1}{3}$ has a unique fixed point on the interval $[-1, 1]$.

It suffices to check if g is a function from $[-1, 1]$ to $[-1, 1]$, and that $|g'(x)| \leqslant k < 1$ for all $x \in [-1, 1]$. If so, g satisfies the premise of the Theorem, and we can immediately conclude that g has a unique fixed point on $[-1, 1]$.

To check if g satisfies that $g : [-1, 1] \to [-1, 1]$, we do optimization. The derivative is $g'(x) = 2x/3$ which means there is only one critical point at $x = 0$. Listing the function values at the critical point and the boundary, we find $g(0) = -1/3$, $g(\pm 1) = 0$, which means $x = 0$ is a global minimum, and $x = \pm 1$ achieve maximum. This suggests that the function values $g(x) \in [-1/3, 0] \subset [-1, 1]$, namely, the function maps the interval $[-1, 1]$ into itself.

To check if g satisfies $|g'(x)| \leqslant k < 1$, we find that

$$|g'(x)| = \left|\frac{2x}{3}\right| \leqslant \frac{2}{3}, \quad \text{for all } x \in (-1, 1).$$

Therefore, by the existence and uniqueness theorem, there is a unique fixed point of $g(x)$ on the interval $[-1, 1]$.

17.3 Fixed Point Iteration Algorithm

Now, though the theorem is useful in the sense that functions with certain properties are guaranteed to have fixed point, it didn't tell us how to find it. In addition, for functions like $g(p) = 3^{-p}$, it is pretty hard to determine the fixed point by hand, that is, solving $p = 3^{-p}$.

Instead, we iterate like the following

$$p_n = g(p_{n-1})$$

by providing an initial point p_0. If the sequence p_n converges to p and g is continuous, then

$$p = \lim_{n\to\infty} p_n = \lim_{n\to\infty} g(p_{n-1}) \overset{\text{continuity}}{=} g\left(\lim_{n\to\infty} p_{n-1}\right) = g(p),$$

and a solution to $x = g(x)$ is obtained. This technique is called **fixed-point**, or **functional iteration**.

17.4 Convergence

An algorithm is never a good one until one proves its convergence. The rate of convergence is the icing on the top.

Theorem 17.2. *Let $g \in C[a,b]$ be such that $g(x) \in [a,b]$, for all $x \in [a,b]$. Suppose in addition that g' exists on (a,b) and there is a constant $0 < k < 1$ such that*

$$|g'(x)| \leqslant k, \quad \text{for all } x \in (a,b).$$

Then for any number $p_0 \in [a,b]$, the sequence defined by

$$p_n = g(p_{n-1}), \quad n \geqslant 1$$

converges to the unique fixed p in $[a,b]$.

Proof. We use a very similar technique as in the uniqueness proof, since we are already provided with the conditions that p is unique. By Mean Value Theorem, there exists ξ_n between p_n and p such that

$$|p_n - p| = |g(p_{n-1}) - g(p)| = |g'(\xi_n)|\,|p_{n-1} - p| \leqslant k\,|p_{n-1} - p|.$$

Now, we repeat the same argument to $|p_{n-1} - p|$ where we further ξ_{n-1} between p_{n-1} and p such that

$$|p_{n-1} - p| = |g(p_{n-2}) - g(p)| = |g'(\xi_{n-1})|\,|p_{n-2} - p| \leqslant k\,|p_{n-2} - p|$$

which then implies, inductively, that

$$|p_n - p| \leqslant k\,|p_{n-1} - p| \leqslant k^2\,|p_{n-2} - p| \leqslant \ldots \leqslant k^n\,|p_0 - p|.$$

Since $0 < k < 1$, we must have $\lim_{n\to\infty} k^n = 0$ and thus

$$\lim_{n\to\infty} |p_n - p| = \lim_{n\to\infty} k^n\,|p_0 - p| = |p_0 - p| \lim_{n\to\infty} k^n = 0,$$

that is, $p_n \to p$ as $n \to \infty$. □

The following corollary provides the error bounds.

Corollary 17.1. *If g satisfies the hypotheses of the Theorem, then bounds for the error involved using p_n to approximate p are given by*

$$|p_n - p| \leqslant k^n \max\{p_0 - a, b - p_0\}$$

and

$$|p_n - p| \leqslant \frac{k^n}{1-k}\,|p_1 - p_0|, \text{for all } n \geqslant 1.$$

Proof. The first inequality is trivial because $p, p_0 \in [a, b]$, so from a step in the proof of the theorem, we readily have

$$|p_n - p| \leqslant k^n |p_0 - p| \leqslant k^n \max \{p_0 - a, b - p_0\},$$

by being conservative with the distance between p_1 and p.

The second inequality is more involved, which we don't present here. The central idea is to study the sequence difference $|p_{n+1} - p_n|$, and then relate it to $|p_m - p_n|$. Then, we bound $|p_m - p_n|$ in terms of $|p_1 - p_0|$. Lastly, we utilize geometric series to express the final inequality. More details are in Theorem 2.5 (of Burden and Faires 9th edition). □

Takeaway 17.1. The most important takeaway from the convergence theorem is that the fixed-point iteration converges the fastest when k is very small, namely, when the derivative $g'(x)$ is very small in magnitude. Therefore, when setting up a fixed-point iteration $p_n = g(p_{n-1})$ for certain root-finding problems $f(p) = 0$, one would be wise to choose a fixed-point map g such that it has the smallest derivative in the search region. This remark should help you interpret the results of HW5.

18 Lecture XVIII: Newton-Raphson Method

In the last section, we saw first how we can turn any root-finding problem to a fixed-point finding problem, and then utilize the fixed-point iteration to find the fixed point. However, fixed-point iteration can become very slow when the iterated function does not have a derivative with small magnitude (one of the sufficient conditions to converge). Newton's method utilizes the slope of the tangent (regardless of its magnitude) to construct linear approximations of the function at each point of the iteration, and finally arrive at the zero of a function without relying on the magnitude of the derivative being smaller than 1.

18.1 Newton's Method

Consider a function $f(x)$ where we know $f(p) = 0$ for $p \in [a, b]$. We start with an initial guess $p_0 \in [a, b]$ such that $f(p_0) \neq 0$. Then, if $f \in C^1[a, b]$, we can compute the equation of the tangent line at p_0,

$$y = f(p_0) + f'(p_0)(x - p_0),$$

i.e., the linearization of f at $x = p_0$.

This tangent line is an approximation of f, which may motivate us to speculate that the x-intercept of this line is close to the zero of f. We see that the x-intercept of the line, p_1, is no other than

$$0 = f(p_0) + f'(p_0)(p_1 - p_0) \implies p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}.$$

Then, we use p_1 as our "guess" for the next approximation. We compute the equation of the tangent line at $x = p_1$ now, and use the x-intercept of this tangent as an approximation p_2. More precisely,

$$p_2 = p_1 - \frac{f(p_1)}{f'(p_1)}.$$

In general, we continue to use the x-intercept of the tangent line at a sequence of points to approximate the true "x-intercept" of f, in the iterative fashion as

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}.$$

This method is called **Newton-Raphson method**, or sometimes in short, **Newton's method**.

The stopping criterion of this method is the sequential absolute error

$$|p_N - p_{N-1}| < \epsilon$$

for some prescribed tolerance level ϵ, or in sequential relative error,

$$\frac{|p_N - p_{N-1}|}{|p_N|} < \epsilon, \quad p_N \neq 0,$$

or in function value

$$|f(p_N)| < \epsilon$$

since we want $f(p_N) \approx 0$.

18.2 Relationship to Fixed-Point Iteration and Convergence

Note that the Newton's method

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$$

is a functional iteration technique with $p_n = g(p_{n-1})$ where

$$g(p_{n-1}) = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{for } n \geqslant 1.$$

Example 18.1. Consider the function $f(x) = \cos(x) - x$. Approximate a root of f using both a fixed-point method and then Newton's method. Which one converges faster?

Solution. The fixed-point method requires a function g_{fix} such that the fixed point of g is a root of f. Clearly, $g_{\text{fix}}(x) = \cos(x)$ works.

Newton's method requires that we identify the iterative function $g_{\text{newt}}(x)$ such that

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

In this problem, we have

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

Thus, to study the convergence framework of Newton's method, we may equivalently study that of fixed-point iterative algorithms.

Theorem 18.1. *Let $f \in C^2[a,b]$. If $p \in (a,b)$ is a root of f such that $f(p) = 0$ and $f'(p) \neq 0$, then there exists a $\delta > 0$ such that Newton's method generates a sequence $\{p_n\}_{n=1}^{\infty}$ converging to p for any initial approximation $p_0 \in [p-\delta, p+\delta]$.*

Proof. Consider the iterative scheme $p_n = g(p_{n-1})$ where

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Then, it is equivalent to show that g has a unique fixed point in $[p-\delta, p+\delta]$. This requires two conditions on g.

1. g maps $[p-\delta, p+\delta]$ to $[p-\delta, p+\delta]$.
2. $|g'(x)| \leqslant k < 1$ for $x \in [p-\delta, p+\delta]$.

The details of the proof are in Theorem 2.6 (Burdon and Faires). Very interesting read with very straightforward computations. □

18.3 Rate of Convergence

Since the Newton's method can be seen as a special case of a fixed point iteration, it is then useful to understand the rate of convergence of fixed point iterations. In the last lecture, we have provided the conditions for the existence and uniqueness of a fixed point, but these results do not inform us how fast the iterations converge to the fixed point.

We first demonstrate the **quadratic** rate of convergence of the Newton's method under mild conditions.

Theorem 18.2 (Quadratic Convergence of Newton's Method)**.** *Let $f \in C^2([a,b])$ with $f'(x) \neq 0$ on $[a,b]$. Let $r \in [a,b]$ be the root such that $f(r) = 0$. Then, given an initial guess p_0 close enough to the root r, the sequence defined by*

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

converges to r at quadratic rate of convergence.

Proof. We consider Newton's method as a fixed point iteration with the function g defined as

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

Let $f(r) = 0$ (and thus $g(r) = r$), where r is the root we wish to converge to. Then, we have the error estimate

$$x_{n+1} - r = g(x_n) - r = g(x_n) - g(r).$$

We expand $g(x_n)$ as a Taylor series about $x = r$, up to the second derivative term,

$$g(x_n) = g(r) + g'(r)(x_n - r) + \frac{g''(\xi)}{2}(x_n - r)^2$$

where $\xi \in [x_n, r]$. Now, observe that the derivative of g satisfies

$$g'(x) = 1 - \frac{[f'(x)]^2 - f(x) f''(x)}{[f'(x)]^2} = \frac{[f'(x)]^2 - [f'(x)]^2 + f(x) f''(x)}{[f'(x)]^2} = \frac{f(x) f''(x)}{[f'(x)]^2}$$

and therefore,

$$g'(r) = \frac{f(r) f''(r)}{[f'(r)]^2} = 0.$$

Thus,

$$g(x_n) = g(r) + \frac{g''(\xi)}{2}(x_n - r)^2.$$

Define the error term $e_n = |x_n - r|$, then we find

$$e_{n+1} = |x_{n+1} - r| = |g(x_n) - g(r)| = \frac{g''(\xi)}{2}(x_n - r)^2 = Ce_n^2$$

where $C = \frac{g''(\xi)}{2}$. In other words, each successive error term is proportional to the square of the previous error, proving that Newton's method has quadratic rate of convergence. □

We have mentioned at the beginning of the section that Newton's method is just a special case of fixed point iteration. As it turns out, the proof above is a special case of a much more general results about fixed point iteration.

Theorem 18.3 (Super Quadratic Convergence of Fixed Point Iteration)**.** *Let p satisfy $p = g(p)$ (a fixed point of g). Suppose that $g'(p) = 0$ and g'' is continuous with $|g''(x)| < M$ on an open interval I containing p. Then there exists $\delta > 0$ such that for $p_0 \in [p - \delta, p + \delta]$, the sequence defined by $p_n = g(p_{n-1})$, when $n \geqslant 1$, converges* ***at least quadratically*** *to p. Moreover, for sufficiently large values of n, we have the error estimate*

$$|p_{n+1} - p| < \frac{M}{2}|p_n - p|^2.$$

The proof is analogous to what we just did, with the help of a vanishing first derivative so that the Taylor expansion loses its first order term. Read Theorem 2.9 in Burden and Faires for details. Complications arise when certain roots are repeated, namely, $f(r) = 0$ AND $f'(r) = 0$ (and possibly vanishing higher order derivatives). Newton's method will suffer a great deal near the root because the slopes are too small to provide substantial advances of the iterates. In fact, the multiplicity of the root affects the rate of convergence of Newton's method (or any fixed point iteration). More analysis is needed.

19 Lecture XIX: Secant Method

19.1 Motivations

In this section, we focus on studying the elementary techniques to numerically find the root of a function $f(x)$, without relying on differentiability (something that can help us in writing, but not numerics).

19.2 Secant Method

In practice, we aren't always blessed with smooth functions. First-order derivatives are commonly approximated by a finite difference instead of being pinpointed by an exact value. Therefore, we consider a more practical version of Newton's method.

We write the derivative as a finite difference

$$f'(p_{n-1}) = \lim_{x \to p_{n-1}} \frac{f(x) - f(p_{n-1})}{x - p_{n-1}}.$$

If $x = p_{n-2}$ is very close to p_{n-1}, then we may write

$$f'(p_{n-1}) \approx \frac{f(p_{n-2}) - f(p_{n-1})}{p_{n-2} - p_{n-1}} = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}.$$

Replacing the derivative in Newton's method, we then have the **Secant Method**, an iteration on

$$p_n = p_{n-1} - f(p_{n-1}) \frac{p_{n-1} - p_{n-2}}{f(p_{n-1}) - f(p_{n-2})}.$$

Note that this iterative scheme is a **second-order recursion**, which means it will requires **two initial guesses** p_0 and p_1. Thanks to the Intermediate Value Theorem, we should choose p_0 and p_1 so that the sign of $f(p_0)$ and $f(p_1)$ are different so that a root is bracketed between p_0 and p_1.

Because of the similarity of the methods, the dynamics of the secant method are also similar to those of Newton's method. Between the two initial points, we connect them to form the first secant line. We find where this line crosses the horizontal axis, named as the third point. Then, connect the second and third point using a secant line, and find its root. Rinse and repeat, we will (hopefully) converge to the root of interest. To guarantee convergence, and to quantify the rate of convergence, we condense them into the following theorem.

19.3 Rate of Convergence of the Secant Method

Theorem 19.1 (Order of Convergence of the Secant Method)**.** *Let $f \in C^2$ on an open interval containing a simple root r, i.e. $f(r) = 0$ and $f'(r) \neq 0$.*

Suppose the secant iteration

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{19.1}$$

is well-defined and that $x_n \to r$.

Define the error $e_n = x_n - r$. Then there exists a constant $C > 0$ such that

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = C,$$

where

$$p = \frac{1+\sqrt{5}}{2}.$$

In particular, the Secant method converges superlinearly with order p.

Proof. **Step 1: Rewrite the iteration in terms of the error.**

Let $x_n = r + e_n$ and $x_{n-1} = r + e_{n-1}$. Then $x_n - x_{n-1} = e_n - e_{n-1}$, and Equation (19.1) becomes

$$e_{n+1} = e_n - f(r+e_n)\frac{e_n - e_{n-1}}{f(r+e_n) - f(r+e_{n-1})}.$$

Step 2: Taylor expansion near the root.

Since $f \in C^2$, Taylor's theorem gives

$$f(r+e) = f(r) + f'(r)e + \frac{1}{2}f''(\xi)e^2,$$

for some ξ between r and $r+e$.

Because $f(r) = 0$, we obtain

$$f(r+e_n) = f'(r)e_n + \frac{1}{2}f''(\xi_n)e_n^2,$$

$$f(r+e_{n-1}) = f'(r)e_{n-1} + \frac{1}{2}f''(\xi_{n-1})e_{n-1}^2.$$

Subtracting,

$$f(r+e_n) - f(r+e_{n-1}) = f'(r)(e_n - e_{n-1}) + \frac{1}{2}f''(\eta_n)(e_n^2 - e_{n-1}^2),$$

for some η_n between x_n and x_{n-1}.

Factor

$$e_n^2 - e_{n-1}^2 = (e_n - e_{n-1})(e_n + e_{n-1}),$$

so that

$$f(r+e_n)-f(r+e_{n-1})=(e_n-e_{n-1})\left[f'(r)+\frac{1}{2}f''(\eta_n)(e_n+e_{n-1})\right].$$

Step 3: Simplify the recurrence.

Substitute the expansions into the update formula. Cancel the common factor $(e_n - e_{n-1})$ and divide numerator and denominator by $f'(r)$:

$$e_{n+1}=e_n-\frac{e_n+\kappa_n e_n^2}{1+\kappa_n(e_n+e_{n-1})},$$

where

$$\kappa_n=\frac{f''(\eta_n)}{2f'(r)}.$$

Since $x_n \to r$, we have $\eta_n \to r$, and thus

$$\kappa_n \to \kappa=\frac{f''(r)}{2f'(r)}.$$

Step 4: Leading-order asymptotics.

Using the expansion

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

and keeping only terms up to total degree 2 in (e_n, e_{n-1}), we obtain

$$e_{n+1}=e_n-\left(e_n+\kappa e_n^2\right)\left(1-\kappa(e_n+e_{n-1})\right)+O(e^3).$$

After cancellation of lower-order terms,

$$e_{n+1}=\kappa e_n e_{n-1}+O(e^3).$$

Thus the dominant relation is

$$e_{n+1}\sim\kappa e_n e_{n-1}. \tag{19.2}$$

Step 5: Determine the order of convergence.

Assume an order $p > 1$ such that

$$e_{n+1}\sim Ce_n^p. \tag{19.3}$$

Reducing the index by one,

$$e_n\sim Ce_{n-1}^p.$$

Now compute e_{n+1} in terms of e_{n-1} in two ways.

From the ansatz:

$$e_{n+1} \sim Ce_n^p \sim C(Ce_{n-1}^p)^p = C^{p+1}e_{n-1}^{p^2}.$$

From Equation (19.2):

$$e_{n+1} \sim \kappa e_n e_{n-1} \sim \kappa(Ce_{n-1}^p)e_{n-1} = \kappa C e_{n-1}^{p+1}.$$

Matching the powers of e_{n-1} gives

$$p^2 = p + 1.$$

Solving,

$$p = \frac{1+\sqrt{5}}{2}.$$

This completes the proof. □

20 Lecture XX: Bracketing Methods

20.1 Bisection: A Binary Search

The idea of the **Bisection Method** originates from the conclusions of the **Intermediate Value Theorem**.

Theorem 20.1. *(Intermediate Value Theorem, IVT) Suppose f is continuous on $[a,b]$ with $f(a)$ and $f(b)$ taking different signs. Then, there exists $p \in (a,b)$ such that $f(p) = 0$.*

Remark 20.1. *This procedure does NOT determine where p is. Nor does the theorem grant uniqueness, i.e., there may be multiple p's that give us $f(p) = 0$. Nonetheless, existence theorem gives us hope of eventually finding the p.*

20.2 The Algorithm

Begin with two candidates $x = a_1$ and $x = b_1$, such that $f(a_1)$ and $f(b_1)$ have different signs. Compute

$$p_1 = a_1 + \frac{b_1 - a_1}{2} = \frac{a_1 + b_1}{2}.$$

- If $f(p_1) = 0$, then we are done.
- If $f(p_1) \neq 0$, then $f(p_1)$ has the same sign as either $f(a_1)$ or $f(b_1)$.
  - If $f(p_1)$ and $f(a_1)$ share the same sign, then we know $p \in (p_1, b_1)$. Set $a_2 = p_1$ and $b_2 = b_1$.
  - If $f(p_1)$ and $f(a_1)$ have different signs, then we know $p \in (a_1, p_1)$. Set $a_2 = a_1$ and $b_2 = p_1$.
- Repeat this procedure for $[a_2, b_2]$ until the function value is sufficiently close to zero.

Example 20.1. Show that $f(x) = x^3 + 4x^2 - 10$ has a root in $[1,2]$ and use the Bisection method to determine an approximation to the root.

Solution. We know $f(1) = -5$ and $f(2) = 14$, so IVT guarantees a root in this interval.

The first step is evaluate the half point of $[1,2]$, namely, $p_1 = f(1.5) = 2.375 > 0$. This implies that we should search $[1, 1.5]$. Then, we go on with the midpoint of this new interval, and find

$$p_2 = f(1.25) = -1.796875 < 0,$$

which prompts the next search interval to be $[1.25, 1.5]$. We find

$$p_3 = f(1.375) = 0.16211 > 0.$$

Continuing this procedure, we eventually find that a_n and b_n become very close to each other. We find

$$p_{13} = 1.365112305$$

which is quite close to the actual root $p = 1.365230013$ (within 10^{-4} error).

Visually speaking, the **Bisection Method** simply halves the search interval until the interval collapses. To numerically carry out the **Bisection Method** is also not difficult (Algorithm 2.1). However, for any iterative algorithm, we want to study the conditions under which the algorithm converges, and if it does, at what rate?

From examples on problems with known solutions, we often observe that certain iterates already produce a good answer (since we know the true solution) but gets discarded because it is still too far from nearby iterates. Therefore, in practice, the **Bisection Method** may become excruciatingly slow since it is throwing out a lot of iterates that may be good enough. However, this method can be used a starter method to help us narrow down the search space, while a secondary method will be utilized for a finer search.

Nonetheless, no matter how slow **Bisection** goes, it **always** converges.

Theorem 20.2. *Suppose that $f \in C[a, b]$ and $f(a) \cdot f(b) < 0$. The Bisection method generates a sequence $\{p_n\}_{n=1}^{\infty}$ approximating a zero p of f with*

$$|p_n - p| \leqslant \frac{b-a}{2^n}, \quad n \geqslant 1.$$

Proof. For each $n \geqslant 1$, we have (halving the interval)

$$b_n - a_n = \frac{1}{2^{n-1}}(b-a), \quad p \in (a_n, b_n),$$

Since $p_n = \frac{1}{2}(a_n + b_n)$ for all $n \geqslant 1$, it follows that p_n and p should not differ by half the interval length, i.e.,

$$|p_n - p| \leqslant \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b-a).$$

Sending $n \to \infty$, we achieve $p_n \to p$. □

Remark 20.2. *The last line of the proof also informs us of the rate of convergence.*

$$|p_n - p| \leqslant (b-a)\frac{1}{2^n}$$

implies that the rate of convergence is $O(2^{-n})$, that is,

$$p_n = p + O\left(2^{-n}\right).$$

Remark 20.3. *The theorem only provides an* ***upper bound*** *for the error, which can be too conservative. In fact, the real error may be far smaller than the upper bound.*

Remark 20.4. *At the same time, one should not be "too impressed" by the exponential convergence here. It is in fact a little deceiving. We demonstrate the true rate of convergence below.*

Suppose ϵ is the given tolerance level, such that we terminate the algorithm when

$$|p_n - p| \leqslant \frac{b-a}{2^n} \leqslant \epsilon$$

Rearranging and solving for n, we have

$$\frac{b-a}{\epsilon} \leqslant 2^n \implies \log_2\left(\frac{b-a}{\epsilon}\right) \leqslant n.$$

This result shows that the number of iterations n required to reduce the error to ϵ is proportional to the logarithm of $\frac{b-a}{\epsilon}$. To put this in a more quantitative context, suppose a user wants to improve from ϵ to $\epsilon/10$, i.e., gaining one more significant digit in his/her approximation. How many iterations are required to achieve this?

$$\log_2\left(\frac{b-a}{\epsilon/10}\right) - \log_2\left(\frac{b-a}{\epsilon}\right) = \log_2\left(\frac{\frac{b-a}{\epsilon/10}}{\frac{b-a}{\epsilon}}\right) = \log_2(10) \approx 3.32.$$

In other words, it takes at least 3 iterations to improve the accuracy of one significant digit. A linear algorithm, i.e., $O(n)$ method (such that $e_{n+1} \leqslant Ce_n$ for $0 < C < 1$ where e_n is the error at the nth step), improves one significant digit per iteration, while the Bisection takes more than 3, making it a **sublinear** algorithm, which converges quite slowly.

Example 20.2. Sketch the graphs of $y = x$ and $y = x^2 - 5$. Use the Bisection method to find the points of intersection.

20.3 Regula Falsi

Both Newton's method and the Secant method are straightforward iterative techniques, where each step follows a simple update formula, and convergence is typically monitored through the sequential error. However, because these methods do not enforce a bracketing interval—where iterates are constrained to an interval guaranteed to contain the root—their updates can sometimes move away from the solution, particularly when applied to functions with small derivatives. The Regula Falsi method, in contrast, maintains bracketing throughout the iteration, ensuring that iterates remain confined within a shrinking region that contains the root. While this improves reliability, it may also slow down convergence in some cases.

20.3.1 Regula Falsi Secant Method

To understand Regula Falsi more precisely, we start with the Secant method, which similarly uses two initial points to approximate the root. Normally, to compute p_3, we use the secant line joining $(p_1, f(p_1))$ and $(p_2, f(p_2))$. However, the Intermediate Value Theorem ensures the existence of a root inside an interval if function values at the endpoints evaluate have different signs. We have three such functions values $f(p_0), f(p_1)$ and $f(p_2)$ for us to decide which secant line we should use to determine p_3.

- If $\text{sgn} f(p_2) \cdot \text{sgn} f(p_1) < 0$, then p_1 and p_2 bracket a root. Thus, we compute p_3 as the x-intercept of the line joining $(p_1, f(p_1))$ and $(p_2, f(p_2))$ (as the naive Secant Method would).
- If not, then we compute p_3 as the x-intercept of line joining $(p_0, f(p_0))$ and $(p_2, f(p_2))$. At this stage we swap the roles of p_0 and p_1 because p_0 is now chosen as one side of the bracket (to maintain the correct bracketing).

Index relabeling is particularly important because it ensures that at every iteration, the method is guided to an interval that definitely contains the root.

20.3.2 Newton-Raphson with Safeguarding

A similar bracketing approach can be applied to Newton's method to enhance its robustness, particularly in ill-conditioned cases. One such approach is called **Newton's method with safeguarding**, namely, given an initial search interval $[a_0, b_0]$,

- If $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \in [a_0, b_0]$, then we accept x_1.
- Otherwise, we perform a Bisection using on $[a_0, b_0]$, and set x_1 as the midpoint of the interval.
- Update the bracketed interval (as in Regula Falsi).

Of course, tracking the bracket requires additional memory, and frequent use of Bisection (as in safeguarding) slows down computation. This method is particularly useful for ill-conditioned root-finding problems, where guaranteed convergence outweighs the additional cost. However, when Newton's method is well-conditioned, we should take full advantage of its rapid quadratic convergence.

21 Lecture XXI: Root-finding in Higher Dimension

21.1 Fixed-Point Methods for Root-finding Problems with Functions of Several Variables

21.1.1 A Review: Functions of Several Independent and Dependent Variables

In the last few classes, we have introduced four critical methods for root-finding of a function with one variable: the Bisection Method, Fixed-point iteration, the Newton-Raphson Method and the Secant Method (and their *regula falsi* variants). In practice, problems tend to be in dimensions more than one, and dimension-reduction techniques can be also highly nontrivial (yet rewarding). We must face the fact that sometimes, we are dealt with a system of **nonlinear** equations,

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0, \\ f_2(x_1, \ldots, x_n) &= 0, \\ &\cdot\cdot \\ &\cdot\cdot \\ &\cdot\cdot \\ f_n(x_1, \ldots, x_n) &= 0, \end{aligned}$$

where each function f_i can be thought of as a mapping

$$f_i : \mathbb{R}^n \to \mathbb{R}.$$

Now, one can put together the f_i also into a vector,

$$\mathbf{F}(x_1, x_2, \ldots, x_n) = (f_1(x_1, \ldots, x_n), f_2(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$$

as each f_i gives an independent output. Thus, the vector-valued function $\mathbf{F}$ is a mapping

$$\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n.$$

The root-finding problem for $\mathbf{F}$ becomes

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}.$$

We say $f_1, f_2, \ldots, f_n$ are the **coordinate functions** of $\mathbf{F}$.

Example 21.1. An ugly set of equations such as

$$\begin{aligned} 3e^{-x_1} + \sin(x_2 x_3) + \frac{1}{2} &= 0 \\ x_1^2 + 4x_2 + x_3^{1/3} &= 0 \\ \cosh(x_1 x_2) + 6x_3 &= 0 \end{aligned}$$

can be condensed in the format

$$\begin{aligned} \mathbf{F}(\mathbf{x}) &= \mathbf{F}(x_1, x_2, x_3) \\ &= (f_1(x_1, x_2, x_3), f_2(x_1, x_2, x_3), f_3(x_1, x_2, x_3))^{\mathrm{T}} \\ &= \left(3e^{-x_1} + \sin(x_2 x_3) + \frac{1}{2}, x_1^2 + 4x_2 + x_3^{1/3}, \cosh(x_1 x_2) + 6x_3\right)^{\mathrm{T}}. \end{aligned}$$

Properties of $\mathbf{F}$, such as limits, continuity, differentiability, etc., require a proper metric. In 1D, this metric is $|x - y|$, namely, the distance between two points. In higher dimensions, the metric is no other than the vector norm (such as l^2-norm) $\|\mathbf{x} - \mathbf{y}\|$. In fact, these properties are well-defined independent of the choice of the norm.

21.1.2 Fixed Points in High Dimensions

We say a function $\mathbf{G} : D \subset \mathbb{R}^n \to \mathbb{R}^n$ has a **fixed point** at $\mathbf{p} \in D$ if $\mathbf{G}(\mathbf{p}) = \mathbf{p}$. It is important to note that the domain and range of $\mathbf{G}$ must have matching dimensions since the fixed point must be of the same dimension to itself.

Recall that in one dimension, in order for $g(x)$ to have a unique fixed point, we require that g maps an interval to itself, and that $|g'(x)| \leqslant k < 1$ for some nonzero k. A similar theorem can be proved for $\mathbf{G}$.

Theorem 21.1. *Let* $D = \left\{(x_1, x_2, \ldots, x_n)^{\mathrm{T}} : a_i \leqslant x_i \leqslant b_i, \quad i = 1, 2, \ldots, n\right\}$ *for some collection of constants* $a_1, a_2, \ldots, a_n$ *and* $b_1, b_2, \ldots, b_n$. *Suppose* $\mathbf{G}$ *is a continuous function from* $D \subset \mathbb{R}^n$ *into* $\mathbb{R}^n$ *with the property that*

1. *(existence)* $\mathbf{G}(\mathbf{x}) \in D$ *whenever* $\mathbf{x} \in D$, *and*
2. *(uniqueness) There exists a constant* $K < 1$ *such that every coordinate function* g_i *satisfies,*

$$\left|\frac{\partial g_i(\mathbf{x})}{\partial x_j}\right| \leqslant \frac{K}{n}, \quad \mathbf{x} \in D,$$

for each $j = 1, 2, \ldots, n$.

Then $\mathbf{G}$ *has a fixed point in* D*, which can be found by the iterative scheme*

$$\mathbf{x}^{(k)} = \mathbf{G}\left(\mathbf{x}^{(k-1)}\right), \quad k \geqslant 1.$$

This sequence converges to $\mathbf{p} \in D$ *and the approximation satisfies error estimates*

$$\left\|\mathbf{x}^{(k)} - \mathbf{p}\right\|_\infty \leqslant \frac{K^k}{1-K}\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|_\infty.$$

Example 21.2. Consider the nonlinear system

$$\begin{aligned} x_1^2 - 10x_1 + x_2^2 + 8 &= 0, \\ x_1 x_2^2 + x_1 - 10x_2 + 8 &= 0. \end{aligned}$$

We turn this into a fixed point iteration using the vector-valued function $\mathbf{G}(x_1, x_2) = (g_1(x_1, x_2), g_2(x_1, x_2))$ by identifying its coordinate functions

$$\begin{aligned} g_1(x_1, x_2) &= \frac{x_1^2 + x_2^2 + 8}{10}, \\ g_2(x_1, x_2) &= \frac{x_1 x_2^2 + x_1 + 8}{10}. \end{aligned}$$

We see that

$$\begin{aligned} x_1^2 - 10x_1 + x_2^2 + 8 = 0 &\iff x_1 = g_1(x_1, x_2), \\ x_1 x_2^2 + x_1 - 10x_2 + 8 = 0 &\iff x_x = g_2(x_1, x_2), \end{aligned}$$

which turns the root-finding problem into a fixed-point finding $\mathbf{x} = \mathbf{G}(\mathbf{x})$.

By some preliminary analysis, we find that $\mathbf{G} : \left[0, \frac{3}{2}\right]^2 \to \left[0, \frac{3}{2}\right]^2$. Indeed, we check the extremes – if $x_1 = x_2 = 0$, we have $0 < g_1(0,0) = 4/5 < 3/2$, and $0 < g_2(0,0) = 4/5 < 3/2$, while if $x_1 = x_2 = 3/2$, $0 < g_1(3/2, 3/2) = 1.35 < 3/2$ and $0 < g_2(3/2, 3/2) = 1.2875 < 1.5$. Thus, existence of a fixed point is guaranteed.

To establish uniqueness, we check the partial derivatives. There are four of them. For $(x_1, x_2) \in \left[0, \frac{3}{2}\right]^2$, we find

$$\begin{aligned} \frac{\partial g_1}{\partial x_1} &= \frac{x_1}{5} \leqslant \frac{3}{10} = \frac{12}{40} \\ \frac{\partial g_1}{\partial x_2} &= \frac{x_2}{5} \leqslant \frac{3}{10} = \frac{12}{40} \\ \frac{\partial g_2}{\partial x_1} &= \frac{x_2^2 + 1}{10} \leqslant \frac{13}{40} \\ \frac{\partial g_2}{\partial x_2} &= \frac{x_1 x_2}{10} \leqslant \frac{9}{40} \end{aligned}$$

So, we may choose $K = \frac{15}{20} < 1$ (this can be chosen even more tightly) so that

$$\left|\frac{\partial g_i(\mathbf{x})}{\partial x_j}\right| \leqslant \frac{K}{n} = \frac{15/20}{2} = \frac{15}{40}.$$

This guarantees that the fixed point is unique.

Then, we perform a functional iteration. To initiate, let's choose $\mathbf{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}\right) = (1,1)$ (better choose something inside D). We find

$$\mathbf{x}^{(1)} = \mathbf{G}\left(\mathbf{x}^{(0)}\right) = \left(g_1(1,1), g_2(1,1)\right) = (1,1).$$

Oops, we found the fixed point in one step. Very lucky. Here is a program that shows convergence if we start elsewhere in D.

21.2 Acceleration using Gauss-Seidel

The functional iteration is of the form

$$\mathbf{x}^{(k)} = \mathbf{G}\left(\mathbf{x}^{(k-1)}\right)$$

where we evaluate the coordinate functions one by one, i.e., for $\mathbf{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}\right)$, the iterates satisfy

$$\begin{aligned} x_1^{(k)} &= g_1\left(x_1^{(k-1)}, x_2^{(k-1)}\right), \\ x_2^{(k)} &= g_2\left(x_1^{(k-1)}, x_2^{(k-1)}\right). \end{aligned}$$

Borrowing from the improvement of **Gauss-Seidel** on **Jacobi**, why don't we do

$$\begin{aligned} x_1^{(k)} &= g_1\left(x_1^{(k-1)}, x_2^{(k-1)}\right), \\ x_2^{(k)} &= g_2\left(\boxed{x_1^{(k)}}, x_2^{(k-1)}\right) \end{aligned}$$

where we use the immediate update within the same iteration? Indeed, see the improvement in convergence from the in-class demonstration.

21.3 Newton's Method in Higher Dimensions

Newton's method in one dimension has a nice geometric interpretation. We approximate the function locally by its tangent line, and thus also approximate the zero of

the function by the zero of the tangent line. In essence, we rely on the linearization formula (first-order Taylor expansion of f)

$$L(x) = f(x_0) + f'(x_0)(x - x_0)$$

and we pose that $L(x_1) = 0$ which yields

$$0 = f(x_0) + f'(x_0)(x_1 - x_0).$$

This, in turn, yields the formula for Newton-Raphson iteration,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

which we push forward the index one by one.

What if $\mathbf{F} = \mathbf{F}(x_1, x_2 \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$, a vector-valued function? Can we motivate a Newton's method in higher dimensions using a similar linearization formula? Maybe we are going too fast. Let's first consider $f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n)$, just a scalar-valued function but with multiple variables. Its first-order Taylor expansion, i.e., linearization, looks like

$$L(\mathbf{x}) = f\left(\mathbf{x}^{(0)}\right) + \left[\frac{\partial f\left(\mathbf{x}^{(0)}\right)}{\partial x_1}, \ldots, \frac{\partial f\left(\mathbf{x}^{(0)}\right)}{\partial x_n}\right] \cdot \left(\mathbf{x} - \mathbf{x}^{(0)}\right) = f\left(\mathbf{x}^{(0)}\right) + \nabla f\left(\mathbf{x}^{(0)}\right) \cdot \left(\mathbf{x} - \mathbf{x}^{(0)}\right).$$

where $\cdot$ means dot product.

Now, treat each coordinate function of $\mathbf{F}$ as f, we simply replace f by f_i and list them in a column vector, namely,

$$\begin{bmatrix} L_1(\mathbf{x}) \\ L_2(\mathbf{x}) \\ . \\ . \\ L_n(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} f_1\left(\mathbf{x}^{(0)}\right) \\ f_2\left(\mathbf{x}^{(0)}\right) \\ . \\ . \\ f_n\left(\mathbf{x}^{(0)}\right) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1\left(\mathbf{x}^{(0)}\right)}{\partial x_1} & \frac{\partial f_1\left(\mathbf{x}^{(0)}\right)}{\partial x_2} & . & . & \frac{\partial f_1\left(\mathbf{x}^{(0)}\right)}{\partial x_n} \\ \frac{\partial f_2\left(\mathbf{x}^{(0)}\right)}{\partial x_1} & \frac{\partial f_2\left(\mathbf{x}^{(0)}\right)}{\partial x_2} & . & . & \frac{\partial f_2\left(\mathbf{x}^{(0)}\right)}{\partial x_n} \\ . & . & . & . & . \\ . & . & . & . & . \\ \frac{\partial f_n\left(\mathbf{x}^{(0)}\right)}{\partial x_1} & \frac{\partial f_n\left(\mathbf{x}^{(0)}\right)}{\partial x_2} & . & . & \frac{\partial f_n\left(\mathbf{x}^{(0)}\right)}{\partial x_n} \end{bmatrix} \left(\mathbf{x} - \mathbf{x}^{(0)}\right)$$

or more compactly,

$$\mathbf{L}(\mathbf{x}) = \mathbf{F}\left(\mathbf{x}^{(0)}\right) + \mathbf{J}_{\mathbf{F}}(\mathbf{x})\left(\mathbf{x} - \mathbf{x}^{(0)}\right).$$

This matrix of partial derivatives is called the **Jacobian** of $\mathbf{F}$, $\mathbf{J}_{\mathbf{F}}(\mathbf{x})$ in short. Now, supposing that some $\mathbf{x}^{(1)}$ gives $\mathbf{L}\left(\mathbf{x}^{(1)}\right) = 0$, we then have

$$\mathbf{F}\left(\mathbf{x}^{(0)}\right) + \mathbf{J}_{\mathbf{F}}\left(\mathbf{x}^{(0)}\right)\left(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right) = \mathbf{0}$$

which implies

$$\mathbf{J_F}\left(\mathbf{x}^{(0)}\right)\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)=-\mathbf{F}\left(\mathbf{x}^{(0)}\right)$$
$$\mathbf{x}^{(1)}-\mathbf{x}^{(0)}=-\left[\mathbf{J_F}\left(\mathbf{x}^{(0)}\right)\right]^{-1}\mathbf{F}\left(\mathbf{x}^{(0)}\right)$$

and ultimately

$$\boxed{\mathbf{x}^{(1)}=\mathbf{x}^{(0)}-\left[\mathbf{J_F}\left(\mathbf{x}^{(0)}\right)\right]^{-1}\mathbf{F}\left(\mathbf{x}^{(0)}\right)},$$

the ever so powerful, Newton-Raphson iteration in high dimension. It requires an initial guess of a vector $\mathbf{x}^{(0)}$.

Compare this to the one-dimensional analog,

$$x^{(1)}=x^{(0)}-\frac{f\left(x^{(0)}\right)}{f'\left(x^{(0)}\right)},$$

the only difference is that the division now is replaced by matrix inversion.

In practice, we actually stop at

$$\mathbf{J_F}\left(\mathbf{x}^{(0)}\right)\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)=-\mathbf{F}\left(\mathbf{x}^{(0)}\right)$$

because this is in the form of

$$A\boldsymbol{y}=\boldsymbol{b}$$

where

$$A=\mathbf{J_F}\left(\mathbf{x}^{(0)}\right),\quad \boldsymbol{y}=\mathbf{x}^{(1)}-\mathbf{x}^{(0)},\quad \boldsymbol{b}=-\mathbf{F}\left(\mathbf{x}^{(0)}\right)$$

while we know what $\mathbf{x}^{(0)}$ is – so solving the linear system for $\boldsymbol{y}$ here informs us the value of $\mathbf{x}^{(1)}$. Meanwhile, $\boldsymbol{y}$ has the interpretation as the **Newton update vector**, just like when we studied the Gauss-Seidel method where we focused on controlling the size of such "update" vector.

21.4 Comparison of Rate of Convergence

Applying the fixed point iteration, and further with Gauss-Seidel acceleration, and lastly the Newton's method, to Example 21.2, we obtain the following convergence table.

iter_count	x_fp		err_fp	x_fpgs		err_fpgs	x_newt		err_newt
1	0.25	0.25	10	0.25	0.25	10	0.25	0.25	10
2	0.8125	0.82656	1.1391	0.8125	0.88633	1.1988	0.87638	0.90126	1.2776
3	0.93434	0.93676	0.12184	0.94457	0.96866	0.13207	0.99595	0.99536	0.23854
4	0.97505	0.97542	0.040714	0.98305	0.99055	0.038479	0.99999	0.99999	0.0092688
5	0.99022	0.99028	0.015167	0.99476	0.99708	0.011705	1	1	1.5462e-05
6	0.99612	0.99613	0.0059003	0.99837	0.99909	0.0036137	1	1	4.4023e-06
7	0.99845	0.99845	0.0023342	0.99949	0.99972	0.001122	1	1	2.7516e-07
8	0.99938	0.99938	0.00092958	0.99984	0.99991	0.00034914	1	1	1.7197e-08
9	0.99975	0.99975	0.00037118	0.99995	0.99997	0.00010874	1	1	1.0748e-09
10	0.9999	0.9999	0.00014837	0.99998	0.99999	3.3881e-05	1	1	6.7177e-11
11	0.99996	0.99996	5.933e-05	1	1	1.0558e-05	1	1	4.1985e-12
12	0.99998	0.99998	2.3729e-05	1	1	3.2903e-06	1	1	2.6235e-13
13	0.99999	0.99999	9.4914e-06	1	1	1.0254e-06	1	1	1.6209e-14
14	1	1	3.7965e-06	1	1	3.1957e-07	1	1	1.2212e-15
15	1	1	1.5186e-06	1	1	9.9593e-08	1	1	2.2204e-16
16	1	1	6.0743e-07	1	1	3.1038e-08	1	1	0
17	1	1	2.4297e-07	1	1	9.6731e-09	1	1	0
18	1	1	9.7189e-08	1	1	3.0146e-09	1	1	0
19	1	1	3.8875e-08	1	1	9.3952e-10	1	1	0
20	1	1	1.555e-08	1	1	2.928e-10	1	1	0
21	1	1	6.2201e-09	1	1	9.1252e-11	1	1	0
22	1	1	2.488e-09	1	1	2.8439e-11	1	1	0
23	1	1	9.9521e-10	1	1	8.8629e-12	1	1	0
24	1	1	3.9808e-10	1	1	2.7622e-12	1	1	0
25	1	1	1.5923e-10	1	1	8.6087e-13	1	1	0
26	1	1	6.3694e-11	1	1	2.6823e-13	1	1	0
27	1	1	2.5477e-11	1	1	8.36e-14	1	1	0
28	1	1	1.0191e-11	1	1	2.609e-14	1	1	0
29	1	1	4.0764e-12	1	1	7.9936e-15	1	1	0
30	1	1	1.6305e-12	1	1	2.5535e-15	1	1	0
31	1	1	6.5226e-13	1	1	8.8818e-16	1	1	0
32	1	1	2.6101e-13	1	1	2.2204e-16	1	1	0
33	1	1	1.0425e-13	1	1	2.2204e-16	1	1	0
34	1	1	4.1744e-14	1	1	0	1	1	0

Figure 1: A comparative study between fixed point iteration with and without Gauss-seidel acceleration, and Newton's method, applied to Example 21.2.

Part V
Function Approximations and Applications

Often in practice, we are provided raw data from experiments and observations, which are inherently **discrete**. In computing and analysis, mathematicians prefer (at least) continuous functions where properties of real numbers can then be utilized. Is there a numerical technique to bridge the **discrete** real world (difficult to analyze) to the **continuous** math realm (with structure)?

22 Lecture XXII: An Introduction to Numerical Optimization

22.1 A Preface to Continuous Optimization Problems

The study of numerical optimization is to algorithmically discover the **global** maximum and minimum values of a function. Below, we focus on first a few results from the perspective of **continuous** optimization problem, as opposed to **discrete** ones (a class in its own right concerning linear programming and semidefinite programming, among many useful but difficult topics).

From Differential Calculus, we have seen statements like the following:

Theorem 22.1 (Extreme Value Theorem)**.** *If f is continuous on $[a, b]$, then f attains its maximum and minimum value, i.e., there exists numbers c and d such that*

$$f(c) \leqslant f(x) \leqslant f(d), \quad \forall x \in [a, b].$$

One may consider this an "existence" theorem of maximum and minimum value of a function. But the theorem does not tell us where exactly the extreme values are, and how we can go about finding them.

Then, we obtain some improvement from differentiability.

Theorem 22.2 (Fermat's Theorem)**.** *Let f be a function on (a, b), and $f(x_0)$ is a local extremum for $x_0 \in (a, b)$. If f is differentiable at x_0, then $f(x_0) = 0$.*

The importance of this Theorem is in the follow-up Corollary:

Corollary 22.1. *The global extrema of a function f on a domain A occur only at boundaries, non-differentiable points, and critical points.*

Fermat's Theorem is better than the Extreme Value Theorem in the sense that we don't have to look for the maximum and minimum everywhere but just at a few places.

There is a high-dimensional analog of this theorem, not stated here for brevity. The general procedure is to set up

1. $\nabla f = 0$, a system of nonlinear equations for the critical points.
2. On the boundary curve $\gamma(t)$ (possibly parametrized by t), we substitute it into the function f and optimize over t, i.e., find the critical points of $\frac{d}{dt} f(\gamma(t)) = 0$. This is **another** possibly nonlinear equation for the critical points on the boundary.
3. List all critical points on the interior and on the boundary, and compare their function values.

To set up a stable and convergent iterative scheme, such as a fixed point iteration or a variant of Newton's method, to solve a high dimensional system of nonlinear equations is NOT easy and is often computationally expensive. But that's NOT the only difficulty. One may obtain more than one solution to the nonlinear system, since all previous theorems are only **existence** guarantees. So, what kind of problems can **guarantee** that the optimum not only exists but is also **unique**?

22.2 A Very Brief Introduction to Convex Optimization

22.2.1 Convex Functions

Where is the minimum of a parabola, if the interval contains the axis of symmetry? Well, duh, $x = -\frac{b}{2a}$, given the parabola is expressed as $f(x) = ax^2 + bx + c$. How can we tell so quickly? Can there be more problems like this?

Yes, this class of problems that guarantees a unique minimum, or, better put as the outcome that any local minimizer is a global minimizer, is called, convex optimization. Let us define a few important terms and their reductions when more conditions are provided.

Definition 22.1 (Convex Function)**.** *A function f is said to be **convex** if*

$$f(\lambda x + (1-\lambda) y) \leqslant \lambda f(x) + (1-\lambda) f(y)$$

for all x, y and $\lambda \in [0,1]$.

Lemma 22.1 (Vector norms are convex functions.)**.** *Any valid norm $\|\cdot\|$ is a convex function.*

Proof. By the triangle inequality and homogeneity of the norm, for any $x, y \in \mathbb{R}^n$ and any $\lambda \in (0,1)$,

$$\|\lambda x + (1-\lambda) y\| \leqslant \|\lambda x\| + \|(1-\lambda) y\| = \lambda \|x\| + (1-\lambda) \|y\|$$

□

In fact, the objective function in the least square problem, namely, $\phi(x) = \|Ax - b\|_2^2$, is also a convex function. It comes from the following fact

Lemma 22.2 (Composition of convex and affine functions)**.** *If $f : \mathbb{R}^n \to \mathbb{R}$ is convex, then for any $A \in \mathbb{R}^{n \times m}$ and any $b \in \mathbb{R}^n$, the function*

$$h(x) = f(Ax + b)$$

is convex.

Proof. By convexity of f, for any $x, y \in \mathbb{R}^m$ and any $\lambda \in (0,1)$,

$$\begin{aligned} h(\lambda x + (1-\lambda) y) &= f(\lambda (Ax + b) + (1-\lambda)(Ay + b)) \\ &\leqslant \lambda f(Ax + b) + (1-\lambda) f(Ay + b) \\ &= \lambda h(x) + (1-\lambda) h(y). \end{aligned}$$

□

With these definitions, we can finally state the punchline theorem

Theorem 22.3 (Local minima are global minima.)**.** *Any local minimum of a convex function is also a global minimum.*

Proof. We prove the claim via contradiction.

Suppose x_{loc} is a local minimum, and x_{glob} is a global minimum such that $f(x_{\text{glob}}) < f(x_{\text{loc}})$. Since x_{loc} is local minimum, there exists some $\delta > 0$ such that $f(x_{\text{loc}}) \leqslant f(x)$ for all $x \in \mathbb{R}^n$ such that $\|x - x_{\text{loc}}\| < \delta$ (namely, you can draw a ball of radius δ around x_{loc} that attains the smallest function value in this neighborhood).

If we choose $\lambda \in (0,1)$ large enough, and define

$$x_\lambda := \lambda x_{\text{loc}} + (1-\lambda) x_{\text{glob}}$$

satisfies $\|x_\lambda - x_{\text{loc}}\|_2 < \delta$ (i.e., find a point on the secant line between x_{loc} and x_{glob} closer to x_{loc}.)

Then,

$$\begin{aligned} f(x_{\text{loc}}) &\leqslant f(x_\lambda) \\ &\leqslant \lambda f(x_{\text{loc}}) + (1-\lambda) f(x_{\text{gloc}}) \quad \text{(convexity of } f\text{)} \\ &< \lambda f(x_{\text{loc}}) + (1-\lambda) f(x_{\text{loc}}) \quad \text{(assumption)} \\ &= f(x_{\text{loc}}) \quad \Longrightarrow \Longleftarrow \quad \text{(Contradiction!!)} \end{aligned}$$

□

22.2.2 The Convex Optimization Problem

A convex optimization problem can be posed as the following

$$\begin{aligned} &\min_{x \in \mathcal{D}} f(x) \\ \text{subject to } &g_i(x) \leqslant 0, \quad i = 1, \ldots, m \text{ (inequality constraints)} \\ &h_j(x) = 0, \quad j = 1, \ldots, n \text{ (equality constraints)} \end{aligned}$$

where f and g_i are all convex functions, and h_j are affine functions (the high dimension analog of linear functions, i.e., $h_j(x) = a_j^{\mathrm{T}} x + b_j$).

Example 22.1. The least square problem

$$\min_{x \in D} \quad \|Ax - b\|_2^2$$

is an **unconstrained** convex optimization problem.

Several caveats that don't typically show up in Calculus I and IV optimization problems.

1. The global minimum are typically **interior points** of the domain, i.e., the critical points. But sometimes the global minimum is on the boundary. So, even if you solved the interior problem using root-finding algorithms, you may not have found the global minimum, which means root-finding is a bit restrictive.

2. At the same time, this highlights that the boundary is quite important, and as soon as we have a multivariable function $z = f(x, y)$ as the objective, the boundary becomes a closed curve or even an infinite cone.

Takeaway 22.1 (Main points)**.**

- Root-finding algorithms are necessary, but they don't always give you the true minimum.

- If the problem is convex, then there is no worry – local minimum is guaranteed to be global minimum.

- But again, not all problems are convex. Are there more robust ways to find the global minimum without relying too much on root-finding algorithms? Next class on Gradient Descent!

23 Lecture XXIII: Gradient Descent and Exact Line Search

Last class, we learned that strictly convex functions have a unique global minimum (uniqueness for you to think about). Therefore, minimizing such a function has a lot of hope. However, setting up $\nabla f = 0$ requires that we take multiple partial derivatives, and to solve the nonlinear system using an iterative method such as the Newton's method requires that we set up

$$\begin{aligned} x_{k+1} &= x_k - \left[\mathbf{J}_{\nabla f}\left(x_k\right)\right]^{-1} \nabla f\left(x_k\right) \\ &= x_k - \left[H_f\right]^{-1} \nabla f\left(x_k\right) \end{aligned}$$

where

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdot & \cdot & \cdot & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

is called the **Hessian** matrix of f, i.e., the matrix of second order partial derivatives.

What if the function f is not twice differentiable? Game over? And even if it is twice differentiable, it is extremely expensive to construct this matrix and evaluate them at the iterates. With all these practicality in mind, we consider the following simple yet efficient method.

23.1 Gradient Descent

For a convex function f, consider an initial point x_0. In order to find the unique minimum x^*, we need to move x_0 to x_1 so that $f\left(x_1\right) < f\left(x_0\right)$, that is, the function value becomes smaller. Notice that this very thought does NOT involve any matrices or derivatives.

In which direction will the function decrease in value? Consider now in general that at the kth step, we want to move our iterates in the direction in which the function decreases in value. This direction must satisfy

$$D_{p_k} f = \nabla f\left(x_k\right) \cdot p_k < 0,$$

that is, the directional derivative is negative.

We also know from Calculus IV that the direction in which the function decreases the fastest is precisely in the direction of $-\nabla f$. Therefore, we construct the following

iterative scheme to find the minimum,

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

which only involves a few evaluations and a derivative, and neither matrices nor matrix inversions.

The α_k's are called step sizes (or learning rate in the ML community). They play an important role in controlling how far the iterates move, just like the SOR puts a relaxation parameter on the Gauss-Seidel direction. There is some art in choosing this step size because too small of a step, the algorithm converges slowly, and yet too large of a step, the algorithm overshoots everytime which may lead to divergence. Let's break down the choice of α_k in further detail, using just knowledge from multivariable calculus.

Before we explore the art of choosing the stepsize, we demonstrate the vanilla gradient descent in an example.

Example 23.1. Consider the function $f(x) = x^2$. Optimize the function on $[-1, 1]$ using gradient descent with $\alpha_k = \alpha$ a constant, and an initial guess $x_0 = 3$.

$$x_{k+1} = x_k - \alpha_k f'(x_k) = x_k - \alpha(2x_k) = (1 - 2\alpha) x_k$$

1. For $\alpha = 0.1$ we have $x_{k+1} = 0.8x_k$, and the first few iterates satisfy

$$\begin{aligned} x_1 &= 0.8x_0 = 0.8 \times 3 = 2.4 \\ x_2 &= 0.8x_1 = 0.8 \times 2.4 = 1.96 \\ x_3 &= 0.8x_2 = 0.8 \times 1.96 = 1.568 \\ &\vdots \end{aligned}$$

   one can tell that $x_k \to 0$ eventually because the prefactor is a fraction. However, the convergence is NOT fast.

2. For $\alpha = 0.4$, we have $x_{k+1} = (0.2) x_k$, which we expect to converge faster since the fraction smaller (see Figure 2a).

3. However, consider a much bigger stepsize $\alpha = 1$, we have

$$x_{k+1} = (1 - 2) x_k = -x_k.$$

Then,

$$\begin{aligned} x_1 &= -x_0 = -3 \\ x_2 &= -x_1 = 3 \\ x_3 &= -x_2 = -3 \\ &\vdots \end{aligned}$$

which does NOT converge. Or try $\alpha = 1.1$ (see Figure 2b)

Example 23.2. Now, what if we want to optimize the same parabola but on $[1, 2]$? We know that the minimum occurs at the boundary point $x = 1$, but does the method help us get there?

The update rule with stepsize α is still the same

$$x_{k+1} = x_k - \alpha\,(2x_k) = (1 - 2\alpha)\,x_k.$$

With $\alpha = 0.1$ and $x_0 = 2$, we have

$$\begin{aligned} x_1 &= 0.8x_0 = 0.8 \times 2 = 1.6 \\ x_2 &= 0.8x_1 = 0.8 \times 1.6 = 1.28 \\ x_3 &= 0.8x_2 = 0.8 \times 1.28 = 1.024 \\ x_4 &= 0.8x_3 = 0.8 \times 1.024 = 0.8192 \end{aligned}$$

Here, we must stop, because the 4th iterates already took us OUTSIDE the domain. However, we trust the gradient descent that it is decreasing function values through all iterates, and clip $x_4 = 1$ as the left endpoint, and conclude that $x = 1$ is the global minimum.

Example 23.3 (A bad step size can destroy convergence)**.** Consider the minimization problem

$$\phi(x) = x^2.$$

Then

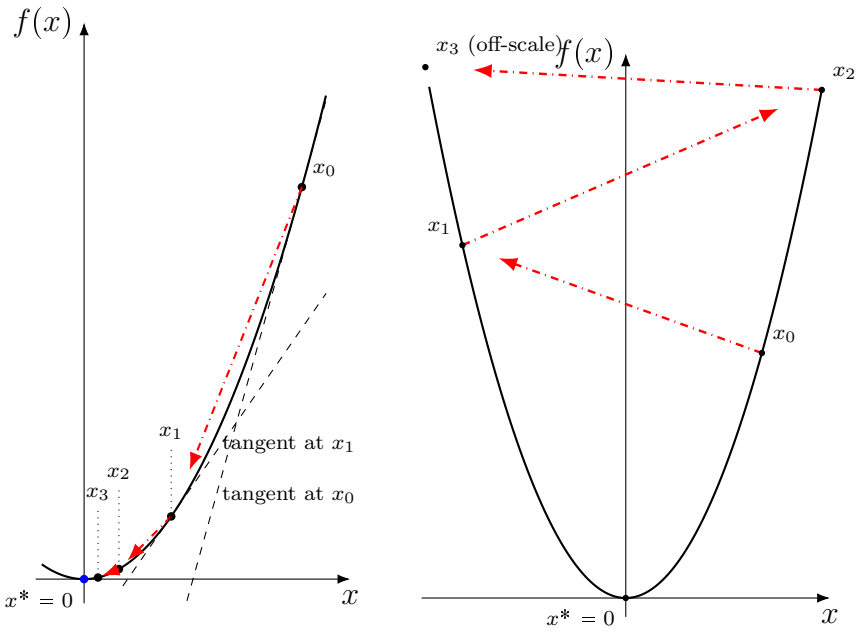
$$\phi'(x) = 2x.$$

Gradient descent with fixed step size α gives

$$x_{k+1} = x_k - \alpha\phi'(x_k) = x_k - 2\alpha x_k = (1 - 2\alpha)x_k.$$

Choose $\alpha = 2$. Then

$$x_{k+1} = -3x_k.$$



(a) $\alpha = 0.4$ (b) $\alpha = 1.1$

Figure 2: An Illustration of Gradient Descent on the Parabola $y = x^2$ with (a) $\alpha = 0.4$ and (b) $\alpha = 1.1$.

Starting from $x_0 = 1.8$, we obtain

$$x_1 = -5.4, \qquad x_2 = 16.2, \qquad x_3 = -48.6.$$

The iterates alternate sign and grow in magnitude. Although we are always moving in the negative gradient direction, the method diverges.

Remark 23.1. *The recurrence*

$$x_{k+1} = (1 - 2\alpha)x_k$$

converges if and only if

$$|1 - 2\alpha| < 1.$$

This gives the admissible range

$$0 < \alpha < 1.$$

Thus, even in the simplest quadratic case, convergence depends entirely on the step size. This example reveals the structural issue.

In one dimension for a quadratic,

$$x_{k+1} = (1 - 2\alpha)x_k.$$

In higher dimensions, for

$$\phi(x) = \tfrac{1}{2}x^T Ax - b^T x,$$

gradient descent becomes

$$x_{k+1} = (I - \alpha A)x_k + \alpha b.$$

Convergence requires

$$\rho(I - \alpha A) < 1,$$

where $\rho(\cdot)$ denotes the spectral radius.

For a general nonlinear function ϕ,

$$x_{k+1} = x_k - \alpha \nabla \phi(x_k),$$

and the allowable range of α is no longer obvious.

Each iteration of gradient descent makes two independent design choices:

$$x_{k+1} = x_k + \alpha_k p_k.$$

- The search direction p_k.
- The step size α_k.

So far we have used

$$p_k = -\nabla \phi(x_k), \qquad \alpha_k = \text{constant}.$$

The divergence above shows that even with a correct descent direction, a poor choice of step size can cause failure. This motivates a more systematic strategy: choose the direction first, then determine the step size by minimizing ϕ along that direction.

23.2 General Line Search

In gradient descent we observed that every iteration has the form

$$x_{k+1} = x_k - \alpha \nabla \phi(x_k).$$

Two independent choices are being made here:

- the **direction** of motion,
- the **step size**.

The divergence example in one dimension made something clear: even if the direction is correct, a poor step size can destroy convergence. Thus, it is conceptually useful to separate these two design decisions.

A unifying framework. A broad class of optimization algorithms can be written in the form

$$x_{k+1} = x_k + \alpha_k p_k,$$

where

- p_k is a chosen search direction,
- $\alpha_k > 0$ is a step length.

This is called a *line search method*, because once p_k is fixed, the new iterate is restricted to lie on the line

$$\{x_k + \alpha p_k : \alpha > 0\}.$$

We then reduce the n–dimensional problem to a one–dimensional one.

Reduction to a one–dimensional problem. Define the scalar function

$$\psi(\alpha) = \phi(x_k + \alpha p_k).$$

Instead of choosing α_k heuristically, we may determine it by minimizing ψ over $\alpha > 0$:

$$\alpha_k = \arg\min_{\alpha > 0} \psi(\alpha).$$

Thus, each iteration consists of:

1. Choose a direction p_k.
2. Minimize ϕ along that direction.
3. Update $x_{k+1} = x_k + \alpha_k p_k$.

The high–dimensional problem is decomposed into:

- a direction–selection mechanism,
- a one–dimensional optimization subproblem.

Descent directions. To ensure decrease of ϕ, we require that p_k be a *descent direction*, meaning

$$\nabla\phi(x_k)^T p_k < 0.$$

Indeed, by Taylor expansion,

$$\phi(x_k + \alpha p_k) = \phi(x_k) + \alpha \nabla\phi(x_k)^T p_k + \mathcal{O}(\alpha^2),$$

so for sufficiently small $\alpha > 0$, the function decreases.

The negative gradient $p_k = -\nabla\phi(x_k)$ is one natural choice, but it is not the only one.

Many familiar methods fit this structure. The following algorithms are all line search methods:

- **Steepest Descent:**
$$p_k = -\nabla\phi(x_k).$$
- **Newton's Method:**
$$p_k = -[\nabla^2\phi(x_k)]^{-1}\nabla\phi(x_k).$$
- **Richardson iteration (quadratic case):**
$$p_k = b - Ax_k.$$

The update formula is identical in all cases:

$$x_{k+1} = x_k + \alpha_k p_k.$$

What distinguishes the methods is how p_k is chosen.

Design trade-off. The line search framework clarifies an important principle:

- A better direction p_k typically requires more information (e.g., second derivatives in Newton's method).
- A more accurate step size α_k typically requires solving an additional one–dimensional optimization problem.

There is no free lunch: robustness and fast convergence come at computational cost.

In the next section we examine the special case of *exact line search*, where the one–dimensional problem is solved precisely.

23.3 Exact Line Search

The first example of the gradient descent applied to the simple parabola is enough evidence that the stepsize α is quite important. Convergence may be too slow or not even possible.

So, how should we choose the stepsize α_k more prudently in order to guarantee convergence and improve rate of convergence?

We actually set this up as **another** optimization problem. The stepsize should be chosen such that the function evaluated at the next iterate is minimized. More precisely, we seek

$$\alpha_k = \arg\min_{\alpha} \left\{ f\left(x_k - \alpha \nabla f\left(x_k\right)\right)\right\}.$$

Define $\varphi(\alpha) = f\left(x_k - \alpha \nabla f\left(x_k\right)\right)$. Then this 1D optimization problem amounts to solve $\varphi'(\alpha) = 0$.

Example 23.4. Let us continue the example on the parabola.

$$\begin{aligned}
\varphi(\alpha) &= f\left(x_k - \alpha f'\left(x_k\right)\right) \\
&= f\left(x_k - 2\alpha x_k\right) \\
&= f\left(\left(1 - 2\alpha\right) x_k\right) \\
&= \left(1 - 2\alpha\right)^2 x_k^2
\end{aligned}$$

and thus

$$\varphi'(\alpha) = -2\left(1 - 2\alpha\right) x_k^2 = 0 \implies \alpha = \frac{1}{2}$$

Thus, gradient descent for the parabola will converge the fastest if

$$x_{k+1} = x_k - \frac{1}{2} f'\left(x_k\right) = x_k - \frac{1}{2}\left(2x_k\right) = 0.$$

This suggests that given **any** initial guess, the iterative method will take you to the minimum in one step.

Example 23.5 (Minimizing a quadratic in high dimension)**.** Consider the function value at the next iterate now in more than one dimension,

$$\varphi(\alpha) = f\left(x_k - \alpha \nabla f\left(x_k\right)\right)$$

so

$$\begin{aligned}
\varphi'(\alpha) &= \nabla f\left(x_k - \alpha \nabla f\left(x_k\right)\right) \cdot \frac{d}{d\alpha}\left(x_k - \alpha \nabla f\left(x_k\right)\right) \\
&= -\nabla f\left(x_k - \alpha \nabla f\left(x_k\right)\right) \cdot \nabla f\left(x_k\right)
\end{aligned}$$

Now, setting

$$\varphi'(\alpha) = -\nabla f\left(x_k - \alpha \nabla f\left(x_k\right)\right) \cdot \nabla f\left(x_k\right) = 0$$

yields a pretty difficult problem unless f is really simple. If f is quadratic,

$$f(x) = \frac{1}{2} x^{\mathrm{T}} A x + b^{\mathrm{T}} x + c$$

where A is positive semi-definite, then

$$\nabla f(x) = Ax + b.$$

The derivative of φ satisfies

$$\begin{aligned}
\varphi'(\alpha) &= -\nabla f(x_k - \alpha \nabla f(x_k)) \cdot \nabla f(x_k) \\
&= -(A(x_k - \alpha \nabla f(x_k)) + b) \cdot \nabla f(x_k) \\
&= -(Ax_k - \alpha A \nabla f(x_k) + b) \cdot \nabla f(x_k) \\
&= -(\nabla f(x_k) - \alpha A \nabla f(x_k)) \cdot \nabla f(x_k) \\
&= -\|\nabla f(x_k)\|_2^2 + \alpha [\nabla f(x_k)]^{\mathrm{T}} A \nabla f(x_k)
\end{aligned}$$

which vanishes when

$$\alpha_k^* = \frac{\|\nabla f(x_k)\|_2^2}{[\nabla f(x_k)]^{\mathrm{T}} A \nabla f(x_k)}.$$

Since A here is the Hessian of the quadratic function, it measures local curvature of the "bowl". The stepsize takes the curvature into account in the denominator (as some kind of a preconditioner) to control the stepsize.

However, in practice, when f is highly nonlinear, solving this additional optimization problem just to determine the optimal stepsize is often infeasible. We need to do more hard work here, to utilize the behaviour of the function near the iterates. What tools are better than Taylor expansions here?!! See more next class on inexact line search inspired by Taylor expansions, and the induced heuristics such as the Armijo backtracking rule.

24 Lecture XXIV: Inexact Line Search and Armijo Backtracking

Last class, we saw the possibility of optimizing the step size (or learning rate) of the gradient descent method, by solving an additional scalar-valued optimization problem,

$$\alpha_k = \arg\min_{\alpha} f\left(x_k - \alpha \nabla f\left(x_k\right)\right)$$

i.e., finding the optimal step size such that the updated function value is minimized. This is known as the **exact line search**.

When f is high dimensional and highly nonlinear, this problem becomes infeasible and requires an approximation instead. The step size is chosen to *approximately* minimize f, or sometimes, just to reduce f "enough".

24.1 Backtracking Line Search

Recall that a general **line search** method has the form

$$x_k = x_{k-1} + \alpha_k p_k$$

where p_k is the descent direction. If it is the gradient, then we set $p_k = \nabla f\left(x_k\right)$. In backtracking line search, we just want the function value at the next update to be small enough (smaller than previous update, of course), which amounts to checking whether

$$f\left(x_k + \alpha_k p_k\right) < f\left(x_k\right) + \gamma \alpha_k \nabla f\left(x_k\right)^{\mathrm{T}} p_k.$$

Now, let's interpret each term. The LHS is the function value at the next iterate. The 1st term on the RHS is the function value at the current iterate. The most important item is the last term: $\nabla f\left(x_k\right)^{\mathrm{T}} p_k$ is precisely the directional derivative, while α_k is the usual step size. The additional factor γ here introduces a margin of safety – the function must decrease **at least** a fraction γ of the predicted decrease. For example, a small γ means we accept small decreases, whereas a larger γ enforces stricter descent.

Define two parameters $\beta \in (0, 1)$ and $\gamma \in (0, 0.5)$. Consider the gradient descent with backtracking algorithm

initialize $\alpha_k = 1$

  while $f\left(x_k - \alpha_k \nabla f\left(x_k\right)\right) > f\left(x_k\right) - \gamma \alpha_k \nabla f\left(x_k\right)^{\mathrm{T}} \nabla f\left(x_k\right)$ (sufficient decrease?)

    do $\alpha_k = \beta \alpha_k$. (take a smaller step)

We should always start strong with a full step size. If we are descending with the largest possible step, then great, continue doing so; however, if we overshoot, then

we should take a smaller step (by a factor of β) and see if we still overshoot. This alrogithm to determine the approximate step size is called *backtracking line search* or *the Armijo rule*. The inequality condition for the while loop is often called *the Armijo condition*.

24.2 Intuition from Taylor Expansions

Let's stare at the Armijo condition for general descent, that is, the condition we wish to achieve at the next iterate,

$$f\left(x_k + \alpha_k p_k\right) \leqslant f\left(x_k\right) + \gamma \alpha_k \nabla f\left(x_k\right)^{\mathrm{T}} p_k.$$

such that the function has sufficient decrease. Hmmm. This looks like some kind of a Taylor expansion. Indeed,

$$f\left(x_k + \alpha p_k\right) \approx f\left(x_k\right) + \alpha \underbrace{\nabla f\left(x_k\right)^{\mathrm{T}} p_k}_{\text{directional derivative}} + O\left(\alpha^2\right).$$

Replacing $p_k = \nabla f\left(x_k\right)$ and scaling the update by γ yields the Armijo condition for gradient descent.

24.3 An Old Example with A New Method

Consider the parabola $f(x) = x^2$ again, and we follow gradient descent

$$x_{k+1} = x_k - \alpha_k f'\left(x_k\right) = \left(1 - 2\alpha\right) x_k.$$

with an initial position $x_0 = 2$. We then declare backtracking parameters: Armijo constant $\gamma = 0.5$, and the reduction factor $\beta = 0.7$.

We begin with $\alpha = 1$, an ambitious full step. We find that

$$f\left(\left(1 - 2\alpha\right) x_0\right) = f\left(-2\right) = 4$$

We check the Armijo condition

$$\begin{aligned} f\left(x_0 - \alpha_0 \nabla f\left(x_0\right)\right) &\overset{?}{\leqslant} f\left(x_0\right) - \gamma \alpha_0 \left\|\nabla f\left(x_0\right)\right\|_2^2 \\ \left(2 - 1 \cdot (2 \cdot 2)\right)^2 &\overset{?}{\leqslant} 2^2 - 0.5 \cdot 1 \cdot |2 \cdot 2|^2 \\ 4 &\overset{?}{\leqslant} -4 \end{aligned}$$

which is NOT satisfied. This means $\alpha = 1$ is too big of a step. We need to shave it off by a factor of β.

Then, take $\alpha \mapsto \alpha \times \beta = 0.7$. We find that

$$\begin{aligned} f\left(x_0 - \alpha_0 \nabla f\left(x_0\right)\right) &\overset{?}{\leqslant} f\left(x_0\right) - \gamma\alpha_0 \left\|\nabla f\left(x_0\right)\right\|_2^2 \\ \left(2 - 0.7 \cdot (2 \cdot 2)\right)^2 &\overset{?}{\leqslant} 2^2 - 0.5 \cdot 0.7 \cdot \left|2 \cdot 2\right|^2 \\ 0.64 &\overset{?}{\leqslant} -1.6 \end{aligned}$$

not quite there, but we see that the gap between the left and right is decreasing.

Take one more! $\alpha \mapsto \alpha \times \beta = 0.49$. We find that

$$\begin{aligned} f\left(x_0 - \alpha_0 \nabla f\left(x_0\right)\right) &\overset{?}{\leqslant} f\left(x_0\right) - \gamma\alpha_0 \left\|\nabla f\left(x_0\right)\right\|_2^2 \\ \left(2 - 0.49 \cdot (2 \cdot 2)\right)^2 &\overset{?}{\leqslant} 2^2 - 0.5 \cdot 0.49 \cdot \left|2 \cdot 2\right|^2 \\ 0.04 &\overset{?}{\leqslant} 0.08 \end{aligned}$$

YES! Finally the inequality is satisfied, so we accept this step size $\alpha_k = 0.49$, and commit to taking the gradient descent using it.

$$\begin{aligned} x_1 &= x_0 - 0.49 \times f'\left(x_0\right) \\ &= 0.04 \end{aligned}$$

Within one step, we are already quite close to the true minimum at $x = 0$.

25 Lecture XXIV: Minibatch Stochastic Gradient Descent with an Application to Logistic Regression

In the previous two lectures, we have explored the method of gradient descent, with exact or inexact line search which aims to fine tune the step size (or learning rate). Regardless of which method one uses, the computation and evaluation of the **full** gradient of f is unavoidable. This procedure becomes significantly more costly when f has a lot of independent variables, which asks for a long and painful gradient computation. So, we wonder if it is really all that necessary to use the full gradient.

Furthermore, in Machine Learning applications, we often try to minimize not just f, but an average of many functions, i.e.,

$$\min_x \quad \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

The naive gradient descent step would then look like

$$x_k = x_{k-1} - \alpha_k \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x_{k-1}).$$

In comparison, the **stochastic** gradient descent or SGD uses

$$x_k = x_{k-1} - \alpha_k \nabla f_{i_k}(x_{k-1}),$$

where the index $i_k \in \{1, \ldots, m\}$ is some (uniformly) randomly chosen index at iteration k.

Noticing that the probability of choosing a particular index is $1/m$, we have

$$\mathbb{E}\left[\nabla f_{i_k}(x)\right] = \frac{1}{m}\sum_{i_k=1}^{m} \nabla f_{i_k}(x) = \nabla f(x)$$

which helps us view the SGD as an **unbiased estimate** of the gradient at each step. In other words, choosing a random direction, is on average, good enough.

Remark 25.1 (Advantages of SGD)**.**

1. *Iteration cost is independent of m, the number of functions.*
2. *Saves a lot of memory since one only needs to access one coordinate of the full gradient.*

One can further select not one but a subset of the indices $I_k \subset \{1, \ldots, m\}$ where $|I_k| = b \ll m$ is called the **minibatch**. We then perform the **minibatch stochastic gradient descent**

$$x_k = x_{k-1} - \alpha_k \frac{1}{b} \sum_{i \in I_k} \nabla f_i \left(x_{k-1} \right),$$

as a tradeoff between memory and fidelity (to the true gradient information).

25.1 An Application to Logistic Regression

25.1.1 The Regression Problem

The logistic regression is a classification method that predicts the probability that a given input x_i belongs to a particular class (typically labeled as $y_i \in 0, 1$). Instead of directly modeling y_i as a linear function of x_i, logistic regression applies the sigmoid function to ensure the output is a probability between 0 and 1.

We assume that given an input $x_i \in \mathbb{R}^n$, the label $y_i \in 0, 1$ follows a Bernoulli distribution

$$P \left(y_i \mid x_i, \beta \right) = \sigma \left(x_i^{\mathrm{T}} \beta \right)^{y_i} \left(1 - \sigma \left(x_i^{\mathrm{T}} \beta \right) \right)^{(1 - y_i)}.$$

For a dataset of n independent observations $\{(x_i, y_i)\}_{i=1}^n$, the likelihood function (joint probability of observing all data points) is

$$L \left(\beta \right) = \prod_{i=1}^{n} \sigma \left(x_i^{\mathrm{T}} \beta \right)^{y_i} \left(1 - \sigma \left(x_i^{\mathrm{T}} \beta \right) \right)^{1 - y_i}.$$

We want to find the set of β that the observed data is most probable, i.e., maximize L over β. The product calls for a negative log-transform, which preserves optimality (due to $\log \left(x \right)$ being monotone) – we just need to minimize instead of maximizing. The **negative log-likelihood** is a **convex** function

$$l \left(\beta \right) = - \sum_{i=1}^{n} \left[y_i \log \left(\sigma \left(x_i^{\mathrm{T}} \beta \right) \right) + (1 - y_i) \log \left(1 - \sigma \left(x_i^{\mathrm{T}} \beta \right) \right) \right].$$

Simplifying using the definition of the sigmoid function and logarithm, and further, divide the sum by n (so that our loss does not depend on the number observables), we arrive at the sample average of the true loss function over all data,

$$\mathcal{L} \left(\beta \right) = \frac{1}{n} \sum_{i=1}^{n} \left[- y_i x_i^{\mathrm{T}} \beta + \log \left(1 + \exp \left(x_i^{\mathrm{T}} \beta \right) \right) \right],$$

where the true loss is

$$\mathbb{E}_{(x,y) \sim \text{all data}} \left[- \left(y \log \left(\sigma \left(x^{\mathrm{T}} \beta \right) \right) + (1 - y) \log \left(1 - \sigma \left(x^{\mathrm{T}} \beta \right) \right) \right) \right].$$

25.1.2 Applying SGD

Given $(x_i, y_i) \in \mathbb{R}^p \times 0, 1$, $i = 1, \ldots, n$, we seek a set of coefficients β that solves the minimization problem

$$\min_{\beta} \quad \frac{1}{n} \sum_{i=1}^{n} \left(-y_i x_i^{\mathrm{T}} \beta + \log \left(1 + \exp \left(x_i^{\mathrm{T}} \beta \right) \right) \right)$$

where we identify the individual function

$$f_i(\beta) = -y_i x_i^{\mathrm{T}} \beta + \log \left(1 + \exp \left(x_i^{\mathrm{T}} \beta \right) \right).$$

The objective function

Coordinate gradient computation results in

$$\nabla f_i(\beta) = -y_i x_i + \underbrace{\frac{\exp \left(x_i^{\mathrm{T}} \beta \right)}{1 + \exp \left(x_i^{\mathrm{T}} \beta \right)}}_{\text{sigmoid function } \sigma(z) = \frac{1}{1+\exp(-z)}} x_i$$

where we recognize

$$\frac{\exp \left(x_i^{\mathrm{T}} \beta \right)}{1 + \exp \left(x_i^{\mathrm{T}} \beta \right)} = \frac{1}{1 + \exp \left(-x_i^{\mathrm{T}} \beta \right)} = \sigma \left(x_i^{\mathrm{T}} \beta \right).$$

Thus, the full gradient is

$$\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} \left(\sigma \left(x_i^{\mathrm{T}} \beta \right) - y_i \right) x_i.$$

When n is moderate, this is feasible, but definitely not when n is huge.