

# CONTENTS

[	Part	1
	1	Chapter 1 Introduction
		1.1 Motivation 3 1.2 Topics 4
	2	Chapter 2 Computer Arithmetic and Computational Errors
		<ul> <li>Numerical Stability 5</li> <li>Floating Point Arithmetic 7</li> <li>Ploating Point Representation 7</li> <li>Computational Errors 11</li> <li>Intuition 11</li> <li>Action 12</li> <li>Cancellation 14</li> </ul>
	3	Chapter 3 Mathematically Conditioning
		3.1 Introduction 19 3.1.1 Problem Statement 19
	4	Chapter 4 Linear systems
		4.1 Introduction 23
		4.1.1 Cramer's Rule 23 4.2 Gaussian Elimination 24
		4.2.1 Intuition 24 4.2.2 Gaussian Elimination 26 4.2.3 Gauss Transforms 28
		4.3 Solving System of Linear Equations 30
		4.3.1 LU Factorization 30 4.4 Gaussian Elimination with Pivoting 33
		4.4.1 Instability of Gaussian Elimination 33
		4.4.2 Partial Pivoting 41 4.4.3 Complete Pivoting 42
		1 0 1

# Chapter 5 Interpolation

	5.1	Introduction $45$ 5.1.1 Definition $45$ 5.1.2 Design Goals for Interpolation $46$ 5.1.3 Interpolation Problems $46$ Solving Interpolation Problems $47$ 5.2.1 Monomial Basis $47$ 5.2.2 Lagrange Basis $49$ 5.2.3 Newton Basis $51$ 5.2.4 Summary $53$	
II	Appendic	ees	55
Bib	liography		57
$\operatorname{Ind}$	ex		59

Part I

Part 1

1

# Introduction

# 1.1 Motivation

 $Math Textbook + Laptop + Coding \stackrel{?}{\Longrightarrow} Compute Accurate Solution$ 

Consider the McLaurin series expansion of the function  $f(x) = e^x$ :

$$e^{x} = 1 + x + \frac{x^{2}}{2!} + \frac{x^{3}}{3!} + \cdots$$

$$= \sum_{n=0}^{\infty} \frac{x^{n}}{n!}$$

The issue is that we cannot compute to infinity. We need to introduce partial sums

$$S_n = \sum_{i=0}^n \frac{x^i}{i!}$$

We could iterate over n until  $|S_n - S_{n-1}| <$ tolerance.

We observe that the running time is dependent on the value of x. We need to find a better way to compute the sum – with more consistent running time.

Using the python program,

- When x = -30, convergence happened after 97 terms, to  $-6.0 \times 10^{-5}$ .
- When x = -40, convergence happened after 124 terms, to approximately  $-5.9 \times 10^{\circ}$ .



Clearly, we have inaccuracy when x = -40, as  $0 < e^x < 1$  for all x < 0. The math textbooks' s techniques does not always provide good computational algorithms.

Course goal:

Show computational algorithms and discuss why they are good.

**Example** ( $e^x$  Better Algorithm). A better algorithm is as follows

- Find k such that  $r = \frac{x}{k}$  exactly with ||r|| < 1.
- Compute  $e^r = e^{x/k}$  using the McLaurin series.
- Then,  $e^x = (e^r)^k$ .



Remark Error due to Catasrophic Cancellation

When we subtract two numbers that are very close to each other, we lose precision.

# 1.2 Topics

- Computer Arithmetic and Computational Errors (Chap. 1)
  - Floating Point Arithmetic
  - Two Concepts
    - The conditioning of a math problem
    - the numerical stability of an algorithm
- Solving Systems of Linear Equations (Chap. 2)
  - Solve Ax = b for x
- $\bullet$  Solving Non-linear Equations (Chap. 5)

Fine x s.t. f(x) = 0 or g(x) = 0 or f(x) = g(x).

- Interpolation (Chap. 7)
  - Given the set of data

$$\{(t_i, y_i)\}_{i=0}^n$$
 or  $\{(t_i, f(t_i))\}_{i=0}^n$ 

come up with a function g(t) that approximates the data.

# 2.1

# **Numerical Stability**

There is only finite space in computer. How would we store  $\pi$ , an irrational number? We can't. We can only store an approximation of  $\pi$ . How does introduction of approximations affect the accuracy of our computations?

Example. Suppose we want to compute the value for the sequence of integrals

$$y_n = \int_0^1 \frac{x^n}{x+5} \, dx$$

for  $n = 0, 1, 2, \dots, 8$ , with 3 decimal digits of accuracy.

There are several properties that I can claim:

- $y_n > 0$  for all n, since the integrand  $\frac{x^n}{x+5} > 0$  for all  $x \in (0,1)$ .
- $y_{n+1} < y_n$  for all n, since the integrand  $\frac{x^{n+1}}{x+5} = x \cdot \frac{x^n}{x+5} < \frac{x^n}{x+5}$  for all  $x \in (0,1)$ .

There is not closed-form solution to this problem.

$$x^{n} = x^{n} \cdot \frac{x+5}{x+5}$$
 for  $x \in (0,1)$ 

$$x^{n} = \frac{x^{n+1}}{x+5} + \frac{5x^{n}}{x+5}$$

$$\int_{0}^{1} x^{n} dx = \int_{0}^{1} \frac{x^{n+1}}{x+5} dx + 5 \int_{0}^{1} \frac{x^{n}}{x+5} dx$$

$$\frac{1}{n+1} x^{n+1} \Big|_{0}^{1} = y_{n+1} + 5y_{n}$$

$$y_{n+1} = \frac{1}{n+1} - 5y_{n}$$

Fortunately,

$$y_0 = \int_0^1 \frac{1}{x+5} dx$$

$$= \ln(x+5) \Big|_0^1$$

$$= \ln 6 - \ln 5$$

$$= \ln \frac{6}{5} \doteq 0.182$$

By the recurrence,

$$y_1 = \frac{1}{1} - 5y_0 \doteq 1 - 5(0.182) = 0.0900$$

$$y_2 = \frac{1}{2} - 5y_1 \doteq 0.5 - 5(0.0900) = 0.0500$$

$$y_3 = \frac{1}{3} - 5y_2 \doteq 0.333 - 5(0.0500) = 0.0830$$

$$y_4 = \frac{1}{4} - 5y_3 \doteq 0.25 - 5(0.0830) = -0.165$$

Clearly, something went wrong. We have a negative value for  $y_4$ , which is impossible. We also have  $y_3 > y_2$ . The problem is that we are using floating point arithmetic, which is not exact. We are losing precision in our calculations.

What if we leave  $y_0$  as an unevaluated term?

$$y_1 = 1 - 5y_0$$

$$y_2 = \frac{1}{2} - 5y_1$$

$$= -\frac{9}{2} + 25y_0$$

$$y_3 = \frac{1}{3} - 5y_2$$

$$= \frac{137}{6} - 125y_0$$

$$y_4 = \frac{1}{4} - 5y_3$$

$$= -\frac{1367}{12} + 625y_0$$

We approximated  $y_0 = \ln \frac{6}{5} \approx 0.182$ . We know that the true value of  $y_0 \in [0.1815, 0.1825]$ . Another way to express  $y_0$  is  $y_0 = 0.182 + E$ , where  $|E| \le 0.0005 = 5 \times 10^{-4}$  is the error in our approximation. Substituting this into the formula for  $y_4$ , we get

$$y_4 = -\frac{1367}{12} + 625(0.182 + E)$$
$$= -113.91\dot{6} + 113.75 + 625E$$
$$= -0.1\dot{6} + 625E$$

where

$$625E \le 625 \times 5 \times 10^{-4} = 0.3125$$

and

$$y_4 < y_0 \doteq 0.182$$

 $\Diamond$ 

so our propagated error is greater than the quantity to compute.

A lesson learned from the previous example is that the math textbook algorithms does not necessarily produce good computational algorithms. This algorithms for computing  $y_n$  is said to be an numerically unstable algorithm, since a small error was magnified by the algorithm. We want the algorithms to be numerically stable.

### **Definition 2.1.1** Numerically Unstable

An algorithm is said to be **numerically unstable** if the error in the output is not bounded by the error in the input.

### Remark

In the previous example, a small error E is magnified by 5 each step.

**Example** (Cont.). We can re-arrange the recurrent relation

$$y_{n+1} = \frac{1}{n-1} - 5y_n$$

$$5y_n = \frac{1}{n+1} - y_{n+1}$$

$$y_{n+1} = \frac{1}{5} \left( \frac{1}{n+1} - y_{n+1} \right)$$

We have bounded the error in the output by the error in the input, but we are at a disadvantage since we are computing backwards. How can we start this recurrent relation?

Recall that

$$y_{100} < y_{99} < \dots < y_0 \doteq 0.182$$

We start by approximating  $y_{100} \doteq 0$ . We know the exact value is  $y_{100} = 0 + \varepsilon$ , where  $0 < \epsilon < 0.182$ . Then,

$$y_{99} = \frac{1}{5} \left( \frac{1}{100} - y_{100} \right)$$

$$= \frac{1}{5} \left( \frac{1}{100} - (0 + \varepsilon) \right)$$

$$= \frac{1}{5} \left( \frac{1}{100} - \left( \frac{1}{500} + \frac{\varepsilon}{5} \right) \right)$$

$$= \frac{1}{500} - \frac{\varepsilon}{5}$$

$$= \dots + \frac{\varepsilon}{25}$$

We observe that the effect of the error is  $\varepsilon$  is diminished by a factor of  $\frac{1}{5}$  each step. This is a numerically stable algorithm. By the time we get to  $y_8$ , we can expect an accurate result.

### **Definition 2.1.2** Numerical Stability

An algorithm is said to be **numerically stable** if the error in the output is bounded by the error in the input.

# 2.2 Floating Point Arithmetic

# 2.2.1 Floating Point Representation

We only have finite space in the computer. This means we cannot store all real numbers, only an approximation of them. We use the **floating point representation** to store real numbers.

### **Definition 2.2.1** Floating Point Representation

A real number x is represented in the form

$$fl(x) = \pm d_0.d_1d_2...d_{p-1} \times \beta^e$$
  $d_0 \neq 0$ 

where m is the significant or mantissa,  $\beta$  is the base, and e is the exponent.

#### Note:

•  $d_i$ 's are bounded by

$$0 \le d_i < \beta$$
.

- p is the precision of the accuracy.
- E is also bounded by

$$L \le E \le U$$

With the floating point representation, we have a finite set of floating point numbers

$$F(\beta, p, L, U) = \{ \pm d_0.d_1d_2...d_{p-1} \times \beta^E : 0 \le d_i < \beta, L \le E \le U \}.$$

# Remark $d_0 \neq 0$

We observe that this representation is not unique. Consider a system

$$F(\beta = 10, p = 5, L = -10, U = 10)$$

and the number 1.23, we have the representation

$$+1.2300 \times 10^{0} \in F(10, 5, -10, 10)$$

but also

$$+0.1230 \times 10^1 \in F(10, 5, -10, 10)$$

and

$$+0.0123 \times 10^2 \in F(10, 5, -10, 10)$$

Thus, we need to choose a unique representation for each number. We can add a rule that **the first digit is not zero**, which normalizes the floating point representation, and makes representations unique.

### Remark Representating Zero

Since  $d_0 \neq 0$ , how can we represent zero? We define another rule

$$fl(0) = 0.000...0 \times \beta^{L-1}$$

where an exponent of L-1 is used to indicate the value is denormalized.

In the early days of computing, different manufacturers used different floating point representations. This made it difficult to write portable code. The IEEE 754 standard [2] was introduced to standardize floating point representations.

	Single Precision	Double Precision
$\beta$	2	2
p	24	53
L	-126	-1022
U	127	1023

### Note:

For single precision, we have

$$E \in \{-126, -125, \dots, 127\}$$

with a cardinality of 254. If we use 8 bits to represent the exponent, we have  $2^8 = 256$  possible

values. We have 2 special values: E = L - 1 = -127 for denormalized numbers, and E = U + 1 = 128 for infinity and NaN (Not a Number).

### Remark Memory Requirements

For single precision, we need 1 + 24 + 8 = 33 bits to represent a number. This is a very weird number of bits. We don't know any computer that uses 33 bits to represent a number.

Turns out we only need 23 bits for the significant, since the first digit in a normalized system is always 1. We can drop the first digit, and use 23 bits for the significant, 8 bits for the exponent, and 1 bit for the sign. This gives us 32 bits, which is a more common number of bits.

### Note: Types in Different Languages

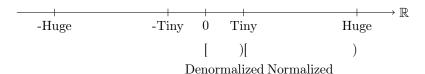
	Single Precision	Double Precision
$\mathbf{C}$	float	double
Python		float
Rust	f32	f64

### Note: Extended Precision and Reduced Precision

For accurate engineering, we may need more precision, called **extended precision** (Quad Precision). This is not part of the IEEE 754 standard, but is available in some systems.

In machine learning systems, we want the opposite – we want to use less precision to save memory and computation time. This is called **reduced precision**, and we often use 16 or 8 bits. This is a topic of active discussion, and the IEEE committee is working on a standard for reduced precision for ML.

Note that the distribution of floating point numbers is not uniform. There are more floating point numbers near zero than near  $\pm \infty$ . This is because the exponent is distributed uniformly, but the significant is not.



$$\begin{array}{lll} & \text{Single} & \text{Double} \\ \text{HUGE} & +1.11\ldots 1\times 2^{127}\approx 3.4\times 10^{38} & +1.11\ldots 1\times 2^{1023}\approx 1.8\times 10^{308} \\ \text{TINY} & +1.00\ldots 0\times 2^{-126}\approx 1.2\times 10^{-38} & +1.00\ldots 0\times 2^{-1022}\approx 2.2\times 10^{-308} \end{array}$$

### Remark

What about  $x \in \mathbb{R}$ , x > HUGE? In this case, we have an overflow, and the computer will return  $fl(x) = +\infty = 1.00 \dots 0 \times \beta^{u+1}$ .

### Remark

What happens if we have  $(+\infty) - (+\infty)$ ? In this case, we have an indeterminate form, and the

computer will return NaN (Not a Number),

$$1.xx...x \times \beta^{u+1}$$

where at least one of the x's is not zero.

## Remark

For  $0 \le x \le \text{TINY}$ , take

$$fl(x) = \begin{cases} 0.00d_i d_{i+1} \dots d_{p-1} \times \beta^{L-1} \\ 0 & \text{eventually} \end{cases}$$

**Example.** Suppose we have

$$F(\beta, p, L, U) = F(10, 3, -10, 10)$$

and

$$x = 3.00 \times 10^5, y = 3.00 \times 10^5, \qquad x, y \in \mathbb{R}$$

Computing

$$z = x^2 + y^2$$

would give

$$z = 1.60 \times 10^{11} + 9.00 \times 10^{10} \notin F(10, 3, -10, 10)$$

and hence fl(z) = +Inf.

What if we instead want to compute

$$z = \sqrt{x^2 + y^2}?$$

Using the standard algorithm, we get +Inf inside the square root, and the computer will return NaN. We need to repair our algorithm.

$$h = \max(|x|,|y|) \times \sqrt{1 + \left(\frac{\min(|x|,|y|)}{\max(|x|,|y|)}\right)^2}$$



### Remark

The libm library in C and the math module in python both have a function called hypot that computes the Euclidean distance. This is a numerically stable algorithm that avoids overflow and underflow.

Note: Inf Could Have Meanings

Suppose you are computing the slope of a line, and you get infinity. This would indicate that the line is parallel to the y-axis. This is a meaningful result, and not an error.

Note: Sometimes, Underflow is OK

Recall the McLaurin series for

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

Suppose we are computing in F(10, 3, -10, 10).

$$y = 1.00 \times 10^{0} + 1.00 \times 10^{-5} + 5.00 \times 10^{-11}$$

The last term would underflow, but that's OK. We can ignore it, since it is negligible.

## 2.3

# **Computational Errors**

## 2.3.1 Intuition

**Example.** We know that  $\sqrt{255} = 15.9687194227...$ , but how should we represent this in our floating point system  $F(\beta, p, L, U) = F(10, 3, -10, 10)$ ?

The easiest solution is chopping / truncation, where we only keep the first p digits. This gives us

$$fl(\sqrt{255}) = 1.59 \times 10^1$$

Another approach is **round to nearest**, which gives us

$$fl(\sqrt{255}) = 1.60 \times 10^1$$



## Remark

Heath called  $x \in \mathbb{R} \to fl(x) \in F$  rounding. This is why we use round-to-nearest to disambiguate.

Note:

We can also round to +Inf or to -Inf

Note: Banker's Rounding

Banker's rounding is a method of rounding that rounds the last digit to the nearest even number, if it is exactly halfway between two numbers. This is the default rounding method in IEEE 754. This method is also known as round to even.

**Example** (Cont.). After rounding, we have error

$$fl(\sqrt{255}) - \sqrt{255} \doteq \begin{cases} -0.0687 & \text{chop} \\ 0.0313 & \text{round-to-nearest} \end{cases}$$



**Definition 2.3.1** Absolute and Relative Error

If  $\tilde{x}$  is an approximation to x, the absolute error in the approximation is

$$|\tilde{x} - x|$$

and the **relative error** is

$$\frac{|\tilde{x} - x|}{|x|} \qquad x \neq 0$$

**Example.** The relative error of the approximation of  $\sqrt{255}$  is

$$\doteq \begin{cases} 4.3 \times 10^{-3} & \text{chop} \\ 1.96 \times 10^{-3} & \text{round-to-nearest} \end{cases}$$



 $\Diamond$ 

### Remark

Absolute error is not sensitive to scale, while relative error is.

**Example.** Let  $x, y \in \mathbb{R}$ , and they are approximated by  $\tilde{x}, \tilde{y} \in F$ .

Consider  $x=100.001, y=0.112, \tilde{x}=100., \tilde{y}=0.113.$  The absolute error is

$$|\tilde{x} - x| = 0.001, \quad |\tilde{y} - y| = 0.001$$

They absolute errors are the same, but this does not meas the two approximations are equally good. The relative error is

$$\frac{|\tilde{x} - x|}{|x|} \doteq 1.0 \times 10^{-5}, \quad \frac{|\tilde{y} - y|}{|y|} \doteq 9.0$$

The relative error for y is much larger than for x, so the approximation for y is worse.

# 2.3.2 Machine Epsilon

If  $x \in \mathbb{R}$  is approximated by  $fl(x) \in F$ , how large can the relative error be?

**Example.** Consider F(10,3,L,U),  $a \in \mathbb{R}$  with  $a = w.xyz \times 10^E$  with  $0 < w \le 9$ ,  $0 \le x,y,z \le 9$ , and  $L \le E \le U$ .

Assume we determine fl(a) using round to nearest,

$$|fl(a) - a| < 0.005 \times 10^{E}$$

The maximum relative error is

$$\max \frac{|fl(a) - a|}{|a|} \le \frac{\max |fl(a) - a|}{\min |a|}$$
$$= \frac{0.005 \times 10^{E}}{1.000 \times 10^{E}}$$
$$= 0.005 = \frac{1}{2} \times 10^{-2}$$



In general, for  $x \in \mathbb{R}$  and  $fl(x) \in F(\beta, p, L, U)$  with round to nearest, we have

$$\frac{|fl(x) - x|}{|x|} \le \frac{1}{2} \times \beta^{1-p}$$

assuming x is representable.

This quantity is called the relative round-off error bound.

### **Definition 2.3.2** Machine Epsilon

The **round-off error bound** or **machine epsilon** is the maximum relative error that can occur when approximating a number,

$$\varepsilon_{\text{machine}} = \frac{1}{2} \times \beta^{1-p}$$

## Remark

In IEEE 754, we have

$$\varepsilon_{\rm machine} = \begin{cases} 2^{-24} \doteq 5.96 \times 10^{-8} & {\rm Single~Precision} \\ 2^{-53} \doteq 1.1 \times 10^{-16} & {\rm Double~Precision} \end{cases}$$

**Example.** Suppose we have F(10, 3, -10, 10),  $x = 1.51 \times 10^8$ , and  $y = 3.71 \times 10^6$ ,  $x, y \in \mathbb{R}$ , fl(x) = x, fl(y) = y.

We want to compute z = x + y. We have

$$z = 1.51 \times 10^8 + 3.71 \times 10^6 = 1.5471 \times 10^8 \notin F.$$



### Remark

Mathematically, addition is not closed in F.

#### Note:

In the CPU, there is extended precision in the floating point unit. This means that the CPU can store more digits than the standard floating point representation. This is useful for intermediate calculations, for example, the  $0.0371 \times 10^8$  in the previous example, but the final result is rounded to the standard representation.

# Example (Cont.).

$$fl(z) = \begin{cases} 1.54 \times 10^8 & \text{chop} \\ 1.55 \times 10^8 & \text{round-to-nearest} \end{cases}$$

We have the relative error

$$\frac{|fl(x+y)-(x+y)|}{|x+y|} = \begin{cases} 0.00459 & \text{chop} \\ 0.00187 & \text{round-to-nearest} \end{cases} \leq \varepsilon_{\text{machine}} = \begin{cases} 0.01 & \text{chop} \\ 0.005 & \text{round-to-nearest} \end{cases}$$



### Remark

For IEEE arithmetic, for  $x, y \in F$  and operation op  $\{+, -, *, /\}$ , we must have

$$\frac{|fl(x \text{ op } y) - (x \text{ op } y)|}{|x \text{ op } y|} \le \varepsilon_{\text{machine}}$$

and the square root of x

$$\frac{|fl(\operatorname{sqrt}(x)) - \sqrt{x}|}{|\sqrt{x}|} \le \varepsilon_{\text{machine}}$$

How does the error accumulates as we perform more operations?

**Example.** • 
$$\exp r1 = \operatorname{sqrt}(2.0) + \log(42.0)$$

• 
$$\exp r1 = \exp(7.0) + \sin(45.0)$$

We have

$$\varepsilon_{\text{expr1}} = \frac{|\text{expr1} - (\sqrt{2} + \ln(42))|}{|\sqrt{2} + \ln(42)|}$$

and

$$\varepsilon_{\mathrm{expr2}} = \frac{|\mathrm{expr2} - (e^7 + \sin(45))|}{|e^7 + \sin(45)|}$$

Suppose we compute

$$\exp r1 + \exp r2$$
  $\exp r1 \times \exp r2$   $\exp r1 \div \exp r2$ 

We know that

- $\varepsilon_{\text{expr1+expr2}} \leq \max(\varepsilon_{\text{expr1}}, \varepsilon_{\text{expr2}}) + |\epsilon_A|$
- $\varepsilon_{\text{expr1} \times \text{expr2}} \le \varepsilon_{\text{expr1}} + \varepsilon_{\text{expr2}} + |\epsilon_B|$
- $\varepsilon_{\text{expr1} \div \text{expr2}} \le \varepsilon_{\text{expr1}} + \varepsilon_{\text{expr2}} + |\epsilon_C|$

The first terms are error due to in exact operands. The last terms are due to the representing the result in floating point, and since they are representation errors, they need to be smaller than the machine epsilon,

$$|\epsilon_A|, |\epsilon_B|, |\epsilon_C| \le \varepsilon_{\text{machine}}$$



### Remark

Error does not grow too quickly, but it does grow. This is why we need to be careful when performing many operations.

What about subtraction? That is a whole different story.

# 2.3.3 Catastrophic Cancellation

**Example.** Suppose we want to find the roots of

$$x^2 + (-320x) + 16 = 0.$$

From math textbooks, we know the quadratic formula

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Using F(10, 4, L, U), we have

$$r_1 = \frac{3.200 \times 10^2 + \sqrt{1.024 \times 10^5 - 6.400 \times 10^1}}{2.000 \times 10^0}$$

$$\stackrel{\cdot}{=} \frac{3.200 \times 10^2 + \sqrt{1.023 \times 10^5}}{2.000 \times 10^0}$$

$$\stackrel{\cdot}{=} \frac{3.200 \times 10^2 + 3.198 \times 10^2}{2.000 \times 10^0}$$

$$= \frac{6.398 \times 10^2}{2.000 \times 10^0}$$

$$= 3.199 \times 10^2$$

1 round off error

Similarly,

$$r_2 = \frac{3.200 \times 10^2 - \sqrt{1.024 \times 10^5 - 6.400 \times 10^1}}{2.000 \times 10^0}$$

$$\stackrel{\cdot}{=} \frac{3.200 \times 10^2 - \sqrt{1.023 \times 10^5}}{2.000 \times 10^0}$$

$$\stackrel{\cdot}{=} \frac{3.200 \times 10^2 - 3.198 \times 10^2}{2.000 \times 10^0}$$

$$= \frac{2.000 \times 10^{-1}}{2.000 \times 10^0}$$

$$= 1.000 \times 10^{-1}$$

You can show that the exact roots are

$$r_1^* \doteq 319.950 \quad r_2^* \doteq 0.050 \qquad r_2^* = 0.05001$$

So the relative errors

$$\varepsilon_{r_1} \doteq 1.6 \times 10^{-4} \le \varepsilon_{\text{machine}} = 5 \times 10^{-4}$$
  $\varepsilon_{r_2} \doteq 1.0$ 

 $\Diamond$ 

## Remark

A machine epsilon of 1 tells us that the result is completely wrong. There is no digits of accuracy in the result

Why would this happen?

$$r_2 = \frac{3.200 \times 10^2 - 3.198 \times 10^2}{2.000 \times 10^0}$$

We know that

- $3.200 \times 10^2$  is exact
- $3.198 \times 10^2$  has 2 rounding errors,  $fl(\sqrt{fl(102400-64)})$
- $2.000 \times 10^{0}$  is exact

We wanted to subtract

$$3.200 \times 10^2 - 3.198$$
xxxxxx  $\times 10^2$ 

We see that the leading terms cancel out, and xxxxxx are insignificant in the intermediate result, but they are significant in the final result. This is known as the catastrophic cancellation.

## **Definition 2.3.3** Catastrophic Cancellation

Catastrophic Cancellation is the loss of significance in the result in subtracting two possibly inexact quantities close in value.

### Remark

To avoid catastrophic cancellation, avoid subtracting nearly equal inexact quantities.

**Example** (Cont.). So how to determine  $r_2$  accurately?

We know that

$$(1x^2 - 320x + 16) = 1 \cdot (x - r_1)(x - r_2)$$

•  $1 \cdot r_1 r_2 = 16$  when x = 0

• 
$$r_2 = \frac{16}{r_1} = \frac{1.600 \times 10^1}{3,199 \times 10^2} = 5.002 \times 10^{-2} < \epsilon_{\text{machine}}$$



### Remark

A better algorithm to compute roots of

$$ax^2 + bx + c = 0$$

is

$$r_1 = \frac{-b - \operatorname{sign}(b)\sqrt{b^2 - 4ac}}{2a} \qquad r_2 = \frac{c}{ar_1}$$

Why there is no catastrophic cancellation in  $b^2 - 4ac$ ?

If there is cancellation, we must have  $b^2 \approx 4ac$ , which means their difference would be a small number, and the square root would be a small number. We later adds it with a much bigger number b, so the small number is insignificant, and the inaccuracy here does not affect the final result.

**Example.** Consider F(10, 4, L, U),  $x = 1, 23 \times 10^{1}$ , and  $y = 2.3xx \times 10^{-2}$ .

$$\begin{aligned} x+y &= 12.34 + 0.023\texttt{xx} \\ &= 12.363\texttt{xx} \xrightarrow{\texttt{round}} fl(x+y) = 1.236 \times 10^1 \end{aligned}$$



### Remark

Given expr1 and expr2, with all positive intermediate results,

$$\varepsilon_{\text{expr1-expr2}} \le \left| \frac{\text{expr1}}{\text{expr1-expr2}} \right| \varepsilon_{\text{expr1}} + \left| \frac{\text{expr2}}{\text{expr1-expr2}} \right| \varepsilon_{\text{expr2}} + \varepsilon_D$$

where  $\varepsilon_D < \varepsilon_{\text{machine}}$  is the representation error.

### Remark

Software try to avoid the subtraction of 2 nearly equal inexact values.

Example. Give an accurate algorithm for computing the value of

$$f(x) = \sqrt{1+x} - 1$$

for x > 0.

The naïve approach would be to compute

$$f = math.sqrt(1 + x) - 1$$

but this would lead to catastrophic cancellation when x is small. The significant of any error in math.sqrt(1+x) is revealed by the subtraction.

A work around is to compute

$$f(x) = (\sqrt{1+x} - 1) \cdot \frac{\sqrt{1+x} + 1}{\sqrt{1+x} + 1}$$
$$= \frac{(1+x) - 1}{\sqrt{1+x} + 1}$$
$$= \frac{x}{\sqrt{1+x} + 1}$$

and since x > 0, there is no catastrophic cancellation. A trade off, however, is that this approach is slower than the naïve approach as we have 1 more operation.

```
FUNCTION F(x)

If x \ge 3 then

f = \text{math.sqrt}(1+x) - 1

ELSE

f = x / (\text{math.sqrt}(1+x) + 1)
```



# Definition 2.3.4 Numerically Stable Algorithm

An algorithm is said to be **numerically stable** if the result it produces is the exact result of a "nearby" problem.

**Example.** Recall computing the root for

$$1 \cdot x^2 + (-320)x + 16 = 0$$

Recall also

$$ax^{2} + bx + c = 0$$
  $a(x - r_{1})(x - r_{2}) = 0$ 

are two ways to express the quadratic.

The quadratic formula gives

$$r_1 \ 3.199 \times 19^2 \quad r_2 = 1.000 \times 10^{-1}$$

For which quadratic equation is this the exact solution?

$$1(x - 3.199 \times 10^{2})(x - 1.000 \times 10^{-1}) = 0$$

which is the same as

$$x^2 - 320x + 31.99 = 0$$

We got the exact roots of a very different problem,  $16 \rightarrow 31.99$  with a relative increment  $\approx 1$ .

The later modifier algorithm gave us

$$r_1 = 3.199 \times 10^2$$
  $r_2 = 5.002 \times 10^{-2}$ 

Fir which quadratic equation is this the exact solution?

$$1(x - 3.199 \times 10^{2})(x - 5.002 \times 10^{-2}) = 0$$

which is the same as

$$x^2 - 319.95002x + 16.001398 = 0$$

The difference

- $\Delta a = 0$
- $\Delta b = 0.04998$  with relative change  $\doteq 1.56 \times 16^4 < \varepsilon_{\rm machine}$
- $\Delta c = 0.01398$  with relative change  $\doteq 8.74 \times 10^{-5} < \varepsilon_{\rm machine}$



# MATHEMATICALLY CONDITIONING

3

# 3.1

# Introduction

## 3.1.1 Problem Statement

Suppose we want to compute f(x) at  $x = \bar{x}$ . Suppose we do not know  $\bar{x}$  exactly, but only an approximation of it,  $\hat{x}$  to  $\bar{x}$  where  $\hat{x} \doteq \bar{x}$ .

# **Definition 3.1.1** Signed Relative Error

The **signed relative error** in approximation of  $\bar{x}$  by  $\hat{x}$  is defined as

$$\delta x = \frac{\hat{x} - \bar{x}}{\bar{x}}.$$

We want to compute  $f(\bar{x})$ , but what we get is  $f(\hat{x})$  – a slightly different problem.

$$f(\hat{x}) = f(\bar{x}(1+\delta x))$$

$$= f(\bar{x} + \bar{x}\delta x)$$

$$= f(\bar{x}) + (\bar{x}\delta x)f'(\bar{x}) + \dots$$

$$\stackrel{:}{=} f(\bar{x}) + (\bar{x}\delta x)f'(\bar{x})$$

$$= f(\bar{x})\left(1 + \frac{\bar{x}f'(\bar{x})}{f(\bar{x})}\delta x\right)$$
Taylor series expansion

The relative error in approximating  $f(\bar{x})$  by  $f(\hat{x})$  is

$$\frac{|f(\hat{x}) - f(\bar{x})|}{|f(\bar{x})|} = \left| \frac{\bar{x}f'(\bar{x})}{f(\bar{x})} \right| |\delta x|.$$

An accurate estimation is when  $\left|\frac{\bar{x}f'(\bar{x})}{f(\bar{x})}\right|$  is small. This is known as the **condition number** of the problem.

### **Definition 3.1.2** Condition Number

The condition number of evaluating f(x) is defined as

$$\kappa_f(x) = \left| \frac{xf'(x)}{f(x)} \right|.$$

### Remark

Note we nave not written any algorithm. The condition number is a pure mathematical concept.

### Remark

When  $\kappa_f(x)$  is small, the problem is well-conditioned. When  $\kappa_f(x)$  is large, the problem is ill-conditioned.

### Remark Rule of Thumb

You loose approximately

$$\log_2 \kappa_f(x)$$

bits of accuracy when you approximate  $f(\bar{x})$  by  $f(\hat{x})$ .

**Example.** Consider  $f(x) = \sqrt{x}$ . Let  $\hat{x} = \bar{x}$ .

How well does  $f(\hat{x})$  approximate  $f(\bar{x})$ ?

Let's look at the conditional number for this function,

$$\kappa_{\sqrt{x}} = \left| \frac{x \frac{d}{dx} \sqrt{x}}{\sqrt{x}} \right|$$

$$= \left| \frac{x \cdot \frac{1}{2} \cdot \frac{1}{\sqrt{x}}}{\sqrt{x}} \right|$$

$$= \frac{1}{2}.$$

Since  $\kappa_{\sqrt{}}(x) < 1$ ,  $\sqrt{\hat{x}}$  is a good approximation to  $\sqrt{\bar{x}}$ . Any error  $\delta x$  in  $\hat{x}$  becomes  $\frac{1}{2}\delta x$  in  $f(\hat{x})$  in the relative sense.

We draw the conclusion that evaluating square roots is a well-conditioned problem. If we have an numerically stable algorithm to compute square roots, we should get an accurate result.

### Remark

If we use a numerically stable algorithm on a well-conditioned problem, we should get an accurate result.

The contra positive is also true: if we get an inaccurate result, then either the problem is ill-conditioned or the algorithm is not numerically stable.

**Example.** Consider  $f(x) = \ln(x)$ . Let  $\hat{x} = \bar{x}$ .

Suppose we want to evaluate  $\ln(0.999)$ , where 0.999 is an intermediate result with some error. We will get  $\ln(0.999) \doteq -1.0005 \times 10^{-3}$ .

Consider a 0.01% relative change to argument,

$$\ln(0.999(1+0.00001)) = -9.0049 \times 10^{-4} = -1.0005 \times 10^{-3}(1+0.09996)$$

We see that the relative error in the result is 9.996%. A small change in the argument results in a large change in the result. This tells us that evaluating logarithms must be an ill-conditioned problem.

$$\kappa_{\ln}(x) = \left| \frac{x \frac{d}{dx} \ln(x)}{\ln(x)} \right|$$
$$= \left| \frac{x \frac{1}{x}}{\ln(x)} \right|$$
$$= \frac{1}{\ln(x)}.$$

3.1. Introduction 21

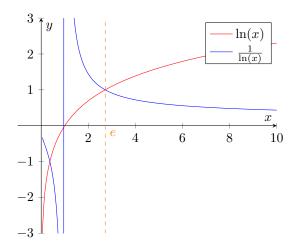
At x = 0.999, we have

$$\kappa_{\ln}(0.999) = \frac{1}{\ln(0.999)} \doteq 999.5.$$

 $\Diamond$ 

For x near 0.999, the effect of a small error is magnified by approximately 1000.

**Example.** Is  $f(x) = \ln(x)$  always ill-conditioned? No.



We see that  $\kappa_{\ln}(x)$  is small for x near 0 and x > e.

What about x near 1? We introduce a new argument w,

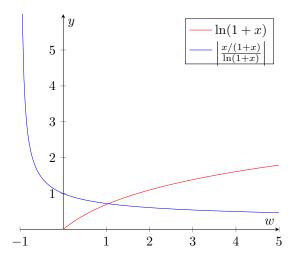
$$\ln(x) = \ln(1+w)$$

Consider  $g(w) = \ln(1+w)$ .

$$\kappa_g(w) = \left| \frac{w \frac{d}{dw} \ln(1+w)}{\ln(1+w)} \right|$$
$$= \left| \frac{w \cdot \frac{1}{1+w}}{\ln(1+w)} \right|$$

We know that f(x) is ill-conditioned for  $x \to 1, w \to 0$ . What is  $\kappa_g(w)$  for  $w \to 0$ ? Using L'Hôpital's Rule,

$$\lim_{w \to 0} \left| \frac{w \cdot \frac{1}{1+w}}{\ln(1+w)} \right| = 1.$$





# Remark

In libm, the C standard math library, the function log1p is used to compute ln(1+x) for small x. This function is numerically stable for small x.

**Example.** How to accurately evaluate

$$f(x) = x = \sqrt{x^2 - 1}$$
 for  $|x| > 1$ 



# LINEAR SYSTEMS

# 4.1

# Introduction

## Definition 4.1.1 Non-Singular Matrix

A coefficient matrix A is **non-singular** if and only if

$$\exists A^{-1}$$
 such that  $AA^{-1} = A^{-1}A = I$ 

### Remark

Some facts

- $A\underline{x} = \underline{b}$  has a **unique** solution if and only if A is non-singular.
- A is non-singular if and only if  $det(A) \neq 0$ .

In the contrast, a singular matrix is one that is not invertible.

## Proposition 4.1.2

If  $\exists z \neq 0$  such that Az = 0, then A is singular.

#### Remark

Suppose exists y such that  $A\underline{y} = \underline{b}$ , then  $y + \alpha z$  solves  $A\underline{x} = \underline{b}$  may not be a solution. For example,

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

has no solution.

We will stick to the case where A is non-singular.

## 4.1.1 Cramer's Rule

### **Definition 4.1.3** Cramer's Rule

For a linear system  $A\underline{x} = \underline{b}$ , the solution is

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where  $A_i$  is the matrix obtained by replacing the *i*th column of A with b.

### Remark

Cramer's Rule is not practical for large systems, as it requires n+1 determinants.

# 4.2

# Gaussian Elimination

Alternatively, we can compute  $A^{-1}$  and then  $x = A^{-1}b$ .

To solve for  $A^{-1}$ , we solve for AY = I, then x = Yb.

For i=1 to n: use some method to solve  $Ay_i=e_i=\begin{pmatrix} 0\\ \vdots\\ 1\\ \vdots\\ 0 \end{pmatrix}$ .

## 4.2.1 Intuition

#### Case I

Suppose A is non-singular diagonal matrix,

$$A = \begin{pmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \end{pmatrix}$$

We must have  $\begin{cases} a_{ij} \neq 0 & \text{if } i = j \\ a_{ij} = 0 & \text{if } i \neq j \end{cases}$ 

Indeed, since A non-singular,  $\det(A) \neq 0$ , and since A diagonal,  $\det(A) = a_1 a_2 \cdots a_n \neq 0$ . None of the diagonal elements can be zero.

Then, for Ax = b, we have  $x_i = b_i/a_{i,i}$ . We can solve this within a single loop.

For 
$$i = 1$$
 to  $n$  do  $x_i = b_i/a_{i,i}$ 

To count the number of operations, we define a new unit of operation, the flop.

### Definition 4.2.1 FLOP

A **FLOP** is a **F**loating-point **L**inear **O**peration. This is a single arithmetic operation on floating-point numbers.

**Example.** For the above algorithm, each iteration require 1 FLOP – a division. Thus, we require n FLOPs to solve the system.

#### Case II

Suppose A is non-singular lower-triangular,

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

We must have  $\begin{cases} a_{ij} = 0 & \text{if } i < j \\ a_{ii} \neq 0 & \text{since } A \text{ non-singular} \end{cases}$ 

- $x_1 = b_1/a_{1,1}$
- $x_2 = (b_2 a_{2,1}x_1)/a_{2,2}$
- $x_3 = (b_3 a_{3,1}x_1 a_{3,2}x_2)/a_{3,3}$
- . :

• 
$$x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j\right) / a_{ii}$$

This algorithm is called **forward substitution**.

```
Function Forward Substitution (A, b)

For i = 1 To n do

sum = 0

For j = 1 To i - 1 do

sum = sum + a_{ij} \cdot x_j

x_i = (b_i - sum)/a_{ii}

Return x
```

We have the operation count

$$\sum_{i=1}^{n} \left[ \left( \sum_{j=1}^{i-1} 2 \right) + 2 \right] = n^2 + \Theta(n) \text{ FLOPs.}$$

### Case III

Suppose A is non-singular upper-triangular,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

We must have  $\begin{cases} a_{ij} = 0 & \text{if } i > j \\ a_{ii} \neq 0 & \text{since $A$ non-singular} \end{cases}$ 

- $\bullet \ x_n = b_n/a_{n,n}$
- $x_{n-1} = (b_{n-1} a_{n-1,n}x_n)/a_{n-1,n-1}$
- $x_{n-2} = (b_{n-2} a_{n-2,n}x_n a_{n-2,n-1}x_{n-1})/a_{n-2,n-2}$
- •

• 
$$x_i = \left(b_i - \sum_{j=i+1}^n a_{ij} x_j\right) / a_{ii}$$

This algorithm is called **backward substitution**.

```
FUNCTION BACKWARDSUBSTITUTION (A, b)
FOR i = n DOWNTO 1 do
sum = 0
FOR j = i + 1 TO n do
sum = sum + a_{ij} \cdot x_j
x_i = (b_i - sum)/a_{ii}
RETURN x
```

The operation count will be the same as for forward substitution,

$$n^2 + \Theta(n)$$
 FLOPs.

### Case IV

What if A is dense? We can use Gaussian elimination to reduce A to a triangular form.

# 4.2.2 Gaussian Elimination

**Example.** Suppose we want to solve the linear system

$$\begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 3 \end{pmatrix}$$

Recall from linear algebra that elementary operations do not change the solution to the system. We can use the elementary row operation

$$R_i = R_i - m \cdot R_i$$

to convert A to an upper-triangular form.

1 
$$R_2 = R_2 - 2 \cdot R_1$$
  $R_3 = R_3 - 4 \cdot R_1$   $R_4 = R_4 - 3 \cdot R_1$ 

$$\begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{pmatrix} \underline{x} = \begin{pmatrix} 1 \\ -1 \\ -5 \\ -6 \end{pmatrix}$$

$$2 R_3 = R_3 - 3 \cdot R_2 R_4 = R_4 - 4 \cdot R_2$$

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

$$3 R_4 = R_4 - R_3$$

$$\begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix} \underline{x} = \begin{pmatrix} 1 \\ -1 \\ -2 \\ 0 \end{pmatrix}$$

We now have a triangular system. We can use backward substitution to solve for x.

$$\underline{x} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

How could we program this algorithm?



```
1: Function Gaussian Elimination (A, b)
        n \leftarrow \operatorname{size}(A)
         ms \leftarrow []
                                                                                                                       ▶ Multipliers
3:
         For i \leftarrow 1 To n-1 do
                                                                                                    ▷ Going down the diagonal
 4:
             FOR j \leftarrow i + 1 To n do
                                                                                                            ▶ Below the diagonal
 5:
                  m \leftarrow A_{i,i}/A_{i,i}
 6:
                  ms[j,i] \leftarrow m
 7:
                  For k \leftarrow i + 1 To n do
 8:
                      A_{j,k} \leftarrow A_{j,k} - m \cdot A_{i,k}
9:
         Return A, ms
10:
```

### Remark

The reason not to update b, but rather to store the multipliers, is that often in practice we have multiple right-hand sides. We can then use the same multipliers to solve for all right-hand sides.

What is the operation count for Gaussian elimination? We consider only the operations on A, since the operations on b are negligible.

- The inner most operation occurs on line 9, with 2 FLOPs.
- They are repeated for k = i + 1 to n,  $\sum_{k=i+1}^{n} 2$  FLOPs.
- There is one more operation on line 6, with 1 FLOP.
- The j loop is repeated for j = i + 1 to n,  $\sum_{j=i+1}^{n} \left(1 + \sum_{k=i+1}^{n} 2\right)$  FLOPs.
- The outer loop is repeated for i=1 to n-1,  $\sum_{i=1}^{n-1} \left(1+\sum_{j=i+1}^{n} \left(1+\sum_{k=i+1}^{n} 2\right)\right)$  FLOPs.

We then simplify

FLOP = 
$$\sum_{i=1}^{n-1} \left( 1 + \sum_{j=i+1}^{n} \left( 1 + \sum_{k=i+1}^{n} 2 \right) \right)$$
  
=  $\sum_{i=1}^{n-1} \left( 1 + \sum_{j=i+1}^{n} \left( 1 + 2(n-i) \right) \right)$   
=  $\sum_{i=1}^{n-1} \left[ (n-i)(1 + 2(n-i)) \right]$ 

Taking m = n - i, we have

FLOP = 
$$\sum_{m=1}^{n-1} m(1+2m)$$
= 
$$\left(\sum_{m=1}^{n-1} m\right) + 2\left(\sum_{m=1}^{n-1} m^2\right)$$
= 
$$\frac{n(n-1)}{2} + 2 \cdot \frac{2n^3 - 3n^2 + n}{6}$$
= 
$$\frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6}$$
= 
$$\frac{2}{3}n^3 + \Theta(n^2) \text{ FLOPs}$$

# 4.2.3 Gauss Transforms

### **Definition 4.2.2** Gauss Transformation Matrix

A  $n \times n$  Gauss transformation matrix  $m_R$  is defined as

$$m_R = I - \underline{m}_r \cdot \underline{e}_r^{\mathsf{T}}$$

where  $m_R \in \mathbb{R}^n$  and

$$\underline{m}_r = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ m_{r+1,r} \\ m_{r+2,r} \\ \vdots \\ m_{n,r} \end{pmatrix} \quad \text{and} \quad \underline{e}_r = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \text{ rth row} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Remark

$$M_{r} = I - m_{r}e_{r}^{\top}$$

$$= I - \begin{pmatrix} 0 & & & & & \\ & \ddots & & & & \\ & & 0 & & & \\ & & m_{r+1,r} & 0 & & \\ & & m_{r+2,r} & 0 & 0 & \\ & & \vdots & & \ddots & \\ & & m_{n,r} & & 0 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -m_{r+1,r} & 1 & & \\ & & -m_{r+2,r} & 0 & 1 & \\ & & \vdots & & \ddots & \\ & & -m_{n,r} & & & 1 \end{pmatrix}$$

Note: Properties of Gauss Transforms

Gauss transforms have nice properties:

1 For indices  $r \leq s$ ,

$$M_r M_s = I - \underline{m}_r \underline{e}_r^{\top} - \underline{m}_s \underline{e}_s^{\top}$$

(not a Gauss transform)

2

$$M_r^{-1} = I + \underline{m}_r \underline{e}_r^{\top}$$

(is a Gauss transform)

3 For any n-vector  $\underline{v} = [v_1, v_2, \dots, v_n]^\top$ ,

$$M_{r}\underline{v} = (I - \underline{m}_{r}\underline{e}_{r}^{\top})\underline{v}$$

$$= \underline{v} - \underline{m}_{r}(\underline{e}_{r}^{\top}\underline{v})$$

$$= \underline{v} - v_{r}\underline{m}_{r}$$

$$\vdots$$

$$v_{r}$$

$$v_{r+1} - v_{r}m_{r+1,r}$$

$$\vdots$$

$$v_{n} - v_{r}m_{n,r}$$

Example. Suppose

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

Choose

$$m_1 = \begin{pmatrix} 0 \\ 2 \\ 4 \\ 3 \end{pmatrix} \implies M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -4 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{pmatrix}$$

We have

$$M_1 A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -4 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{pmatrix}$$

• Choose

$$m_2 = \begin{pmatrix} 0 \\ 0 \\ 3 \\ 4 \end{pmatrix} \implies M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & -4 & 0 & 1 \end{pmatrix}$$

We have

$$M_2 M_1 A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & -4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 4 \end{pmatrix}$$

• Choose

$$m_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \implies M_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

We have

$$M_3 M_2 M_1 A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

We end up with an upper-triangular matrix

$$U = M_3 M_3 M_1 A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

This also implies

$$A = (M_3 M_2 M_1)^{-1} U = M_1^{-1} M_2^{-1} M_3^{-1} U$$

$$= (I + \underline{m}_1 \underline{e}_1^{\top}) (I + \underline{m}_2 \underline{e}_2^{\top}) (I + \underline{m}_3 \underline{e}_3^{\top}) U$$
 by property 2
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{pmatrix} U$$

and we have an anti lower-triangular matrix

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{pmatrix}$$

We can think of Gaussian Elimination as factoring A into LU, where L is lower-triangular and U is upper-triangular.

Verify:

$$LU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

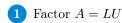


## 4.3

# Solving System of Linear Equations

## 4.3.1 LU Factorization

How to solve  $A\underline{x} = \underline{b}$ ?



The operation cont is

$$\frac{2}{3}n^3 + \Theta(n^2) \text{ FLOPs}$$

Then, we have

$$L(U\underline{x}) = \underline{b}$$

Let  $y = U\underline{x}$ , we have an triangular system

$$Ly = \underline{b}$$

2 Solve  $Ly = \underline{b}$  for y using forward substitution.

The operation count is

$$n^2 + \Theta(n)$$
 FLOPs

Then, we have

$$U\underline{x} = y$$

3 Solve  $U\underline{x} = \underline{y}$  for  $\underline{x}$  using backward substitution.

The operation count is

$$n^2 + \Theta(n)$$
 FLOPs

The total operation count is

$$n^2 + \Theta(n)$$
 FLOPs

4 The total cost is

$$\frac{2}{3}n^3 + \Theta(n^2)$$
 FLOPs

### Remark

Now, suppose we are given a new problem

$$A\underline{z} = \underline{c}$$
 for  $\underline{z}$ 

where A is the same as before, but  $\underline{c}$  is different.

Note that the output from step 1 depends solely on A, and we do not need to recompute it. We skip the most expensive step, and use the same L and U to solve for  $\underline{z}$  using forward and backward substitution.

- Solve  $L\underline{d} = \underline{c}$  for  $\underline{d}$  using forward substitution
- Solve  $U\underline{z} = \underline{d}$  for  $\underline{z}$  using backward substitution

The total new cost is

$$2n^2 + \Theta(n)$$
 FLOPs

### Remark

How to reduce memory usage?

Note that

- L is unit lower triangular (diagonal is all 1s)
- ullet U is upper triangular, and
- A is dense.

We can store L and U in a single matrix, for example,

$$\begin{pmatrix}
2 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
4 & 3 & 2 & 2 \\
3 & 4 & 1 & 2
\end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

#### Edge-case

**Example.** Suppose we want to compute the LU factorization of

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The algorithm fails from 0 division

$$m_{1,1} = \frac{1}{0}$$

However, we could perform a row exchange  $R_1 \leftrightarrow R_2$  to get

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

which is now solvable.



We can use permutation matrix to keep track row exchanges. The permutation matrix for the i-th permutation interchanging rows  $i, r_i$  is the identity matrix with rows  $i, r_i$  swapped.

#### Remark

This is also the identity with **columns**  $i, r_i$  interchanged.

**Example.** Consider  $3 \times 3$  matrices.

$$P_{2,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

is the permutation matrix that swaps rows 2 and 3.

We verify that

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} a & b & c \\ g & h & i \\ d & e & f \end{pmatrix}$$

What if we perform right-multiplication, i.e.  $AP_{2,3}$ ?

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} a & c & b \\ d & f & e \\ g & i & h \end{pmatrix}$$

which swaps columns 2 and 3.



#### Remark

We can use permutation matrices to keep track of row exchanges in LU factorization. Suppose we have a permutation matrix  $P_{i,r_i}$  and a matrix A.

- $P_{i,r_i}A$  swaps  $rows i, r_i$  in A.
- $AP_{i,r_i}$  swaps columns  $i, r_i$  in A.

Note: Facts

Note that

$$P_{i,r_i}^{-1} = P_{i,r_i}^{\top} = P_{r_i,i}.$$

Example. Suppose after two steps of gaussian elimination, we have

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ 0 & x_{22} & x_{23} & x_{24} & x_{25} \\ 0 & 0 & x_{33} & x_{34} & x_{35} \\ 0 & 0 & x_{43} & x_{44} & x_{45} \\ 0 & 0 & x_{53} & x_{54} & x_{55} \end{pmatrix}$$

- If  $x_{33} \neq 0$ , no interchanging is required, and  $P_{i,r_i} = I$ .
- However, if  $x_{33} = 0$ , we need to get the zero off the diagonal We only consider the submatrix

$$\begin{pmatrix} x_{33} & x_{34} & x_{35} \\ x_{43} & x_{44} & x_{45} \\ x_{53} & x_{54} & x_{54} \end{pmatrix}$$

since the rest of the matrix is already in upper-triangular form.



The gaussian elimination factorization becomes

$$M_{n-1}P_{n-1,r_{n-1}}\cdots M_2P_{2,r_2}M_1P_{1,r_1}A=U$$

we can re-arrange to isolate the permutations from the eliminations.

# 4.4 Gaussian Elimination with Pivoting

## 4.4.1 Instability of Gaussian Elimination

**Example.** Consider a  $3 \times 3$  case

$$M_2 P_2 M_1 P_1 A = U$$

with

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}$$

We have

$$P_2 M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

which is not a permutation matrix.

We can then write

$$P_2 M_1 P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}$$

Since  $P_2^{-1} = P_2$ , we have

$$P_2M_1P_2P_2 = P_2M_1$$

This allows us to re-write the factorization as

$$U = M_2 P_2 M_1 P_1 A$$
  
=  $M_2 P_2 M_1 P_2 P_2 P_1 A$   
=  $M_2 \hat{M}_1 P_2 P_1 A$ 

where  $\hat{M}_1 = P_2 M_1 P_2$ 

 $\Diamond$ 

For larger systems, we generalize

$$U = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots P_3M_2P_2M_1P_1A$$

$$= M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots P_3M_2\hat{M}_1P_2P_1A$$

$$= M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots P_3\hat{M}_2P_2P_1A$$

$$\vdots$$

$$= \hat{M}_{n-1}P_{n-1}P_{n-2}\cdots P_3P_2P_1A$$

A where  $\hat{M}_1 = P_2 M_1 P_2$ where  $\hat{M}_2 = P_3 M_2 \hat{M}_1 P_3$ 

We re-label as

$$L^{-1}PA = U$$

where

- $L^{-1} = \hat{M}_{n-1}$
- $P = P_{n-1}P_{n-2}\cdots P_3P_2P_1$

Our algorithm now becomes

- 1 Find P, L, U such that PA = LU
- 2 Since  $A\underline{x} = \underline{b}$ ,  $PA\underline{x} = P\underline{b}$
- 3 Solve  $LU\underline{x} = P\underline{b}$  for  $\underline{x}$ 
  - a Solve  $L\underline{y} = P\underline{b}$  for  $\underline{y}$  using forward substitution
  - **b** Solve  $U\underline{x} = y$  for  $\underline{x}$  using backward substitution

#### Remark

We ask the following questions

1 How to deduce a simple expression for  $P^{-1}$ 

$$\begin{split} P^{-1} &= (P_{n-1}P_{n-2}\cdots P_3P_2P_1)^{-1} \\ &= P_1^{-1}P_2^{-1}\cdots P_{n-2}^{-1}P_{n-1}^{-1} \\ &= P_1^{\top}P_2^{\top}\cdots P_{n-2}^{\top}P_{n-1}^{\top} \\ &= (P_{n-1}P_{n-2}\cdots P_3P_2P_1)^{\top} \\ &= P^{\top} \end{split}$$

- 2 How to store P?
  - Naïvely, we need  $n^2$  entries to store P.
  - However, since  $P = P^{\top}$ , we only need n entries, storying one non-zero entry per row.

Define the *n*-vector  $\underline{p}$ , where  $\underline{p}_i$  is the column index of the non-zero entry in row *i*. For example, we could store

$$\underline{p} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} \qquad \text{for} \qquad P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

In Gaussian Elimination, keep track of interchange using the pivot vector piv.

- 1 Start with the pivot vector  $piv = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ n \end{pmatrix}$  representing I
- 2 For  $R_i \leftrightarrow R_j$ , swap  $piv_i$  and  $piv_j$
- 3 If at the end of GE, piv = (4, 1, 3, 2), then we have

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

#### Note: Implementation Issue

Interchanging  $R_i \leftrightarrow R_{r_i}$ , we do not want to mechanically interchanging via

$$a_{i,j} \leftrightarrow a_{r_i,j}$$

Instead, we could do **indirect addressing** via the pivot vector *piv*.

Suppose we swap piv(i) and piv(j), then a reference to a(i,j) becomes a reference to a(piv(i),j).

#### A Note on Matrix Multiplication

#### Order of Multiplication Matters

Suppose we multiple  $A \in \mathbb{R}^{m \times n}$  by  $B \in \mathbb{R}^{n \times p}$  to get C = AB, we have  $C \in \mathbb{R}^{m \times p}$  with

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

The operation count would be

$$2mnp$$
 FLOPs

Now, suppose we compute

$$A_{n\times 1} = Q_{n\times n} R_{n\times n} S_{n\times 1}$$

• If we compute (QR)S, the operation count is

$$2n^3 + 2n^2$$
 FLOPs

• However, if we compute Q(RS), the operation count is

$$4n^2$$
 FLOPs

This is roughly half the cost of (QR)S.

#### Inverse is Costly

Suppose we have a non-singular M, and we want to compute

$$x = M^{-1}y + \underline{z}$$

- Find  $M^{-1}$  by solving MW = IThis step will cost  $\frac{2}{3}n^3 + \Theta(n^2)$  FLOPs for factorization, and  $n\left(2n^2 + \Theta(n)\right)$  FLOPs for solving. This requires a total of  $\frac{8}{3}n^3 + \Theta(n^2)$  FLOPs.
- Compute  $M^{-1}y$ , which requires  $2n^2$  FLOPs.
- We add  $\underline{z}$ , which requires n FLOPs.

We could alternatively convert this problem into solving a linear system. Let  $\underline{c} = M^{-1}y$ , then

$$M\underline{c} = y$$

We could find  $\underline{c}$  with  $\frac{2}{3}n^3 + \Theta(n^2) + 2n^2 + \Theta(n) = \frac{2}{3}n^3 + 3n^2 + \Theta(n)$  FLOPs. This is about 4 times cheaper than the direct method.

#### Accessing the Accuracy of a Solution

Suppose we want to solve Ax = b with

$$A = \begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \qquad \text{and} \qquad \underline{b} = \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}$$

Suppose also we have two solutions

$$\hat{\underline{x}} = \begin{pmatrix} 0.341 \\ -0.087 \end{pmatrix} \quad \text{and} \quad \tilde{\underline{x}} = \begin{pmatrix} 0.999 \\ -1.001 \end{pmatrix}$$

Which answer is closer to the solution? We could compute the residual and compare the norms of the residuals.

#### Definition 4.4.1 Residual

The **residual** of a solution  $\underline{x}$  to  $A\underline{x} = \underline{b}$  is

$$\underline{r} = \underline{b} - A\underline{x}$$

It measures how much the solution does not satisfy the equation.

With the two solutions, we have

$$\frac{\hat{r}}{\hat{r}} = \underline{b} - A\underline{\hat{x}} 
= \begin{bmatrix} 0.000 & 001 \\ 0 \end{bmatrix} \qquad \frac{\underline{x}}{\hat{x}} = \underline{b} - A\underline{\tilde{x}} 
= \begin{bmatrix} 0.001 & 343 \\ 0.001 & 572 \end{bmatrix}$$

We conclude that  $\hat{x}$  is a better solution.

However, if we take a look at the true solution

$$x^* = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

we have

$$\hat{e} = \hat{x} - x^*$$
  $\tilde{e} = \tilde{x} - x^*$   $= \begin{bmatrix} -0.659 \\ 0.913 \end{bmatrix}$   $= \begin{bmatrix} -0.001 \\ 0.001 \end{bmatrix}$ 

and we see that  $\tilde{x}$  is a better solution.

#### Remark

A small residual need not imply an accurate solution.

We try to explain the phenomenon by looking at the condition number of the matrix A. Suppose we solve  $A\underline{x} = \underline{b}$ , the exact solution is  $\underline{x}^*$ , and we have an approximate solution  $\underline{\bar{x}}$ .

- Define  $\underline{\bar{b}} = A\underline{\bar{x}}$  the right hand side for solving  $A\underline{x} = \underline{b}$  that has solution  $\underline{\bar{x}}$ .
- Define  $\underline{\bar{r}} = \underline{b} \underline{\bar{b}}$  the residual for the approximate solution.

Since  $A\underline{\bar{x}} = \underline{\bar{b}}$  and  $A\underline{x}^* = \underline{b}$ , we have

$$A(\underline{\bar{x}} - \underline{x}^*) = \underline{\bar{b}} - \underline{b} = -\underline{\bar{r}}$$

We are interested in the error

$$\bar{e} = \bar{x} - x^*$$

We have

$$A\underline{\bar{e}} = -\underline{\bar{r}}$$
$$\bar{e} = -A^{-1}\bar{r}$$

To convert the absolute error into a scalar quantity, we use the norm

$$\|\underline{\bar{e}}\| = \left\| -A^{-1}\underline{\bar{r}} \right\| \le \left\| A^{-1} \right\| \cdot \|\underline{\bar{r}}\|$$

Moreover, we know that

$$A\underline{x}^* = \underline{b}$$
$$||A\underline{x}^*|| = ||\underline{b}||$$

and so

$$||A|| \cdot ||\underline{x}^*|| \ge ||||A\underline{x}^*||| = ||\underline{b}||$$

Thus, we have

$$\begin{split} &\frac{\|\underline{\bar{x}} - \underline{x}^*\|}{\|A\| \cdot \|\underline{x}^*\|} \leq \frac{\left\|A^{-1}\right\| \cdot \|\underline{\bar{r}}\|}{\|\bar{b}\|} \\ &\frac{\|\underline{\bar{x}} - \underline{x}^*\|}{\|\underline{x}^*\|} \leq \left(\|A\| \cdot \|A^{-1}\|\right) \frac{\|\underline{\bar{r}}\|}{\|\underline{b}\|} \\ &\leq \|A\| \|A^{-1}\| \end{split}$$

 $||A|||A^{-1}||$  is the **normwise relative error** in computed solution, compared to the **nromwise relative residual** (relative to right hand side  $\underline{b}$ ).

The quantity

$$||A|||A^{-1}||$$

is called the **condition number** of the matrix A.

#### Remark

If the relative residual in the computed solution is small, and the condition number of the matrix is small, then the relative error in the computed solution is small, and the solution is accurate.

#### Remark Vector Norme

$$\|\cdot\|:\mathbb{R}^n\to\mathbb{R}^{\geq 0}$$

• Positive semi-definite

$$\|\underline{x}\| \ge 0$$
 and  $\|\underline{x}\| = 0$  if and only if  $\underline{x} = \underline{0}$ 

• Homogeneous

$$\|\alpha \underline{x}\| = |\alpha| \|\underline{x}\|$$
 for all  $\alpha \in \mathbb{R}$ 

• Triangle Inequality

For all 
$$\underline{x}, \underline{y} \in \mathbb{R}^n$$
,  $||\underline{x} + \underline{y}|| \le ||\underline{x}|| + ||\underline{y}||$ 

There are many norms, but we will focus on the p-norms.

• The Manhattan norm (1-norm)

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

• The Euclidean norm (2-norm)

$$\|\underline{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$$

• The infinity norm

$$\|\underline{x}\|_{\infty} = \max_{1 \le i \le n} |x_i|$$

#### Remark Matrix Norms

$$\|\cdot\|:\mathbb{R}^{n\times n}\to\mathbb{R}^{\geq 0}$$

• Positive semi-definite

$$\|A\| \geq 0$$
 and  $\|A\| = 0$  if and only if  $A = 0$ 

Homogeneous

$$\|\alpha A\| = |\alpha| \|A\|$$
 for all  $\alpha \in \mathbb{R}$ 

• Triangle Inequality

For all 
$$A, B \in \mathbb{R}^{n \times n}$$
,  $||A + B|| \le ||A|| + ||B||$ 

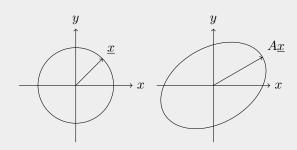
• Cauchy-Schwarz Inequality

For all 
$$A, B \in \mathbb{R}^{n \times n}$$
,  $||AB|| \le ||A|| ||B||$ 

We use induced norms for matrices, for example,

$$||A||_1 = \max_{||\underline{x}||=1} ||A\underline{x}||$$

That is, consider all vectors in the unit circle and apply A to them. The maximum value of the norm of the resulting vectors is the induced matrix norm.



Induced norms are

• The 1-norm

$$\begin{split} \|A\|_1 &= \max_{\|\underline{x}\|_1 = 1} \|A\underline{x}\|_1 \\ &= \max \text{absolute column sum} \\ &= \max_{i \leq j \leq n} \left(\sum_{i=1}^n |a_{i,j}|\right) \end{split}$$

• The 2-norm

$$||A||_2 = \sqrt{\lambda_{\max}(A^\top A)}$$

• The infinity norm

$$\begin{split} \|A\|_{\infty} &= \max_{\|\underline{x}\|_{\infty} = 1} \|A\underline{x}\|_{\infty} \\ &= \max \text{absolute row sum} \\ &= \max_{1 \leq i \leq n} \left( \sum_{j=1}^{n} |a_{i,j}| \right) \end{split}$$

Another useful norm is the **Frobenius norm** 

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

**Example.** We revisit the example, where

$$A = \begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix}$$

We have

$$||A||_{\infty} = 1.572$$

We have defined

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

but we did not define which norm to use. We could have different condition numbers for different norms.

$$\kappa_{\infty}(A) = ||A||_{\infty} ||A^{-1}||_{\infty}$$

From linear algebra, we know that for a  $2 \times 2$  matrix

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

we have its inverse

$$M^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Thus, we have

$$A^{-1} = \frac{1}{0.780 \cdot 0.659 - 0.563 \cdot 0.913} \begin{pmatrix} 0.659 & -0.563 \\ -0.913 & 0.780 \end{pmatrix}$$

so

$$||A^{-1}||_{\infty} \approx 1.693 \times 10^6$$

and hence the conditional number

$$\kappa_{\infty}(A) \approx 2.66 \times 10^6$$

This shows that solving the system is ill-conditioned.

The relative error in our compute solution is bounded above by

 $2.66 \times 10^6$  · relative residual in computed solution



How small can the condition number be? There is a relation between condition number and near-singularity.

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

$$\geq ||AA^{-1}||$$

$$= ||I||$$

$$= 1$$

The condition number is bounded below by 1, and the closer it is to 1, the better conditioned the matrix is.

**Example.** We consider another system

$$Ax = b$$

where

$$\begin{pmatrix} 1.0 \times 10^{-4} & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

The exact LU factorization yields

$$L = \begin{pmatrix} 1 & 0 \\ 1.0 \times 10^{-4} & 1 \end{pmatrix}$$
 and  $U = \begin{pmatrix} 1.0 \times 10^{-4} & 1 \\ 0 & -9.999 \end{pmatrix}$ 

With forward solve,

$$L\underline{y} = \underline{b}$$

$$y_1 = 1$$

$$y_2 = -9.998$$

and then with backward solve,

$$U\underline{x} = \underline{y}$$

$$x_2 = \frac{9998}{9999}$$

$$x_1 = \frac{1000}{9999}$$

 $\Diamond$ 

Now, suppose we get

$$\bar{L} = \begin{pmatrix} 1.00 \times 10^0 & 0\\ 1.00 \times 10^4 & 1.00 \times 10^0 \end{pmatrix} \quad \text{and} \quad \bar{U} = \begin{pmatrix} 1.00 \times 10^{-4} & 1.00 \times 10^0\\ 0 & 1.00 \times 10^4 \end{pmatrix}$$

and we compute

$$\bar{y}_1 = 1.00 \times 10^1$$

$$\bar{y}_2 = fl\left(\frac{2 - 1.00 \times 10^4}{1.00 \times 10^0}\right) = -1.00 \times 10^4$$
 error

and then

$$\bar{x}_2 = 1.00 \times 10^0$$
  
 $\bar{x}_1 = 0.00 \times 10^0$ 

While  $\bar{x}_2$  is accurate,  $\bar{x}_1$  is not.

We first check if the matrix is ill-conditioned. We have

$$||A||_{\infty} = 2$$

and

$$||A^{-1}||_{\infty} = \left| \frac{1}{1.0 \times 10^{-4} - 1} \right| 2 = 2$$

so

$$\kappa_{\infty}(A) = 4$$

The large error is not due to ill-conditioning.

We observe that

$$x_1 = \frac{1.00 - 1.00}{1.0 \times 10^{-4}}$$

and we see that we have catastrophic cancellation here.

We also observe that

$$A = LU$$

where

$$\kappa(L) \approx 1.0 \times 10^8$$
 and  $\kappa(U) \approx 1.0 \times 10^8$ 

so in fact we replaced a well conditioned problem with an ill-conditioned one.

Note: Fact

Whenever multipliers are large (because diagonal elements are relatively small) a loss of accuracy can be observed.

## 4.4.2 Partial Pivoting

A solution to this issue is to try reordering the equations.

**Example** (Cont.). We perform row exchange  $R_1 \leftrightarrow \mathbb{R}_2$ ,

$$\begin{pmatrix} 1 & 1 \\ 1 \times 10^{-4} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

and we obtain

$$L = \begin{pmatrix} 1.00 & 0 \\ 1.00 \times 10^{-4} & 1.00 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 1,00 & 1.00 \\ 0 & fl(1-1 \times 10^{-4}) = 1.00 \end{pmatrix}$$

so

$$\kappa(L) = 1$$
 and  $\kappa(U) = 4$ 

This way, we have a well-conditioned problem.

We now have

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 2.00 \\ 1.00 \end{pmatrix} \quad \text{and then} \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1.00 \\ 1.00 \end{pmatrix}$$

which is the correct floating point representation of the exact solution.

#### $\Diamond$

#### **Definition 4.4.2** Pivot Position

The **pivot position** is the position in the matrix that is used to eliminate the other elements in the column.

Suppose we have a matrix

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}$$

We interchange  $R_i, R_{r_i}$  using  $P_{i,r_i}$  to move small value off of diagonal. Choose interchange rows  $r_i$  such that  $||a_{r_i,i}|| \ge ||a-j,i||$  for all j from i to n.

- Guarantees multipliers are  $\geq 1$  in size
- Do rows interchanges to ensure multipliers are  $\leq 1$

Note: Fact

In practice, when row interchanges are used, the Gaussian Elimination algorithm is **stable**. and guarantees approximate solutions with relative errors  $\frac{\|\underline{r}\|}{\|\underline{b}\|} \approx C_n \cdot \varepsilon_{\text{mach}}$  where  $C_n$  is a small constant dependent on n.

Thus,  $\frac{\|\underline{x} - \underline{x}^*\|}{\|\underline{x} - \underline{x}^*\|} < \kappa(A) \cdot \varepsilon_{\text{mod}}$ 

$$\frac{\|\underline{x} - \underline{x}^*\|}{\|\underline{x}^*\|} \le \kappa(A) \cdot \varepsilon_{\text{mach}} \approx \kappa(A) \cdot C_n \cdot \varepsilon_{\text{mach}}$$

#### Remark Rule of Thumb

Then number of bits of accuracy in the computed solution is approximately  $P - \lceil \log_{\beta} \kappa(A) \rceil$ , where P is the precision of the floating point system.

Remark Terms

Row interchanges, row pivoting, and partial pivoting are all the same thing.

## 4.4.3 Complete Pivoting

In row pivoting, we only search the columns, while in complete pivoting, we search the entire uneliminated submatrix

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}$$

Suppose  $|a_{s,t}| > |a_{i,j}|$  for al i, j, s, t in the un-eliminated submatrix. In complete pivoting, we interchange to put  $a_{s,t}$  in the pivot position. Our factorization becomes

$$PAQ = LU$$

where

- P is a permutation matrix for row interchanges
- ullet Q is a permutation matrix for column interchanges

$$A\underline{x} = \underline{b}$$
 
$$PAQQ^{\top}\underline{x} = P\underline{b}$$
 
$$LUQ^{\top}\underline{x} = P\underline{b}$$

#### Remark Considerations

Complete pivoting is not always necessary, is more computationally expensive than partial pivoting since we need to search in 2D space instead of 1D space.

*Proof.* WTS  $||A||_{\infty}$  is the max absolute row sum of A.

Let x be a unit vector, then

$$||A||_{\infty} = \max_{||\underline{x}||=1} ||A\underline{x}||_{\infty}$$

and

$$\|\underline{x}\|_{\infty} = \max_{1 \le i \le n} |x_i|$$

Let  $\underline{x}$  be such that  $\|\underline{x}\| = 1$  and write  $A = [\underline{a}_1, \underline{a}_2, \dots, \underline{a}_n]$  where  $\underline{a}_i$  is the *i*-th column of A. Then,

$$A\underline{x} = x_1\underline{a}_1 + x_2\underline{a}_2 + \dots + x_n\underline{a}_n = \sum_{j=1}^n x_j\underline{a}_j$$

and

$$(A\underline{x})_i = \sum_{j=1}^n x_j a_{i,j}$$

so

$$||A||_{\infty} = \max_{1 \le i \le n} |(Ax)_i| = \max_{1 \le i \le n} \left| \sum_{j=1}^n x_j a_{i,j} \right|$$

How to maximize for  $\|\underline{x}\|_{\infty} = 1$ ?

For each  $1 \le i \le n$ , maximize by taking  $x_j = 1 \cdot \text{sign}(a_{ij})$ . Then,

$$x_j a_{ij} = |a_{ij}|$$

and so

$$\left| \sum_{j=1}^{n} x_j a_{ij} \right| = \sum_{j=1}^{n} |a_{ij}|$$

and so

$$||A||_{\infty} = \max_{1 \le i \le n} \sum_{i=1}^{n} |a_{ij}|$$

which is the max absolute row sum of A.

# INTERPOLATION

#### 5.1

## Introduction

#### 5.1.1 Definition

Example. Suppose we are given the following population data

year	1940	1950	1960	1970	1980	1990	2000	2010
Population (M)	132	151	179	203	226	249	281	308

We mey ask the question: what was the population in 1965? We can use **interpolation** to estimate the population in 1985.

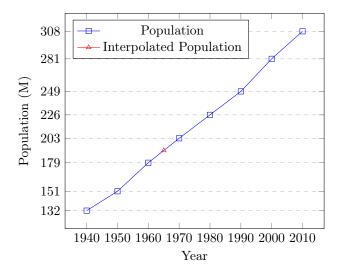


Figure 5.1: Population data and interpolated population in 1965

We may also ask the question: what will the population be in 2020? This will be an **extrapolation** problem.

#### Definition 5.1.1

Given a set of m data points  $\{(t_i, y_i)\}_{i=1}^m$  with  $t_1 < t_2 < \cdots < t_m$ , the **interpolation problem** is to find a function g(t) from a specific class of functions that

$$g(t_i) = y_i$$
 for all  $i = 1, 2, ..., m$ 

#### Remark

Another use of interpolation is to approximate a function f(t) by a simpler function g(t) that is easier to work with.

**Example.** Suppose we have a function

$$\operatorname{erf} = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

and we want to approximate it with a polynomial.

- Evaluate erf at m points,  $\{(x_i, e_i)\}_{i=1}^m$
- Interpolate the points with a polynomial p(x) such that  $p(x_i) = e_i$  for all i = 1, 2, ..., m
- Use the interpolant p(x) in place of erf



### 5.1.2 Design Goals for Interpolation

- 1 Easy to evaluate quick, accurate
- 2 Gives accurate approximations for  $t \neq t_i$
- 3 Can integrate and differentiate easily

#### Remark Caveats

Interpolations come with some caveats.

- Interpolations are not uniques.
- Not all interpolations have nice properties
- Interpolations does not always give the right solution choice

#### Note:

We assume accurate data has the form

$$\{(t_i, y_i)\}_{i=1}^m$$
 or  $\{t_i, f(t_i)\}_{i=1}^m$ 

with

$$t_1 < t_2 < \dots < t_m$$

## 5.1.3 Interpolation Problems

**Example.** Suppose we want to construct a linear fit for the data

$$\{(-1,2),(1,1)\}$$

Define 
$$P_1(t)=b+mt$$
. We have  $\begin{cases} P_1(-1)=2\\ P_1(1)=1 \end{cases} \implies \begin{cases} b+m(-1)=2\\ b+m(1)=1 \end{cases}$  which yields the solution  $b=-\frac{1}{2} \quad b=\frac{3}{3} \quad \text{so} \quad P_1(t)=\frac{3}{2}-\frac{1}{2}t$ 



**Example.** Suppose we want to construct a quadratic fit for the data

$$\{(-1,2),(1,1),(2,1)\}$$

Define 
$$P_2(t) = a + bt + ct^2$$
. We have 
$$\begin{cases} P_2(-1) = 2 \\ P_2(1) = 1 \end{cases} \implies \begin{cases} a + b(-1) + c(-1)^2 = 2 \\ a + b(1) + c(1)^2 = 1 \\ a + b(2) + c(2)^2 = 1 \end{cases}$$
 which yields

$$P_2(t) = \frac{4}{3} - \frac{1}{2}t + \frac{1}{6}t^2$$



Note:

We choose polynomial interpolants

$$P_{m-1}(t) = \sum_{i=1}^{m} c_i t^{i-1}$$

because they are easy to evaluate. The operation count is approximately

$$3n$$
 FLOPs

#### Theorem 5.1.2 Horner's Rule

We can re-write our polynomial as

$$P_{m-1}(t) = c_1 + t(c_2 + t(c_3 + t(\cdots + t(c_{m-1} + c_m t))))$$

Note:

Horner's methods provide some advantages. The operation count reduces to

$$2m$$
FLOPs

and it's easier to integrate and differentiate.

#### Theorem 5.1.3

For a set of points

$$\{(t_i, y_i)\}_{i=1}^m,$$

there exists a unique polynomial of degree at less than m that interpolates the data.

## 5.2

## **Solving Interpolation Problems**

#### 5.2.1 Monomial Basis

Given a set of points

$$S = \{(t_i, y_i)\}_{i=1}^m$$

we find the unique polynomial

$$P_{m-1}(t) = \sum_{i=1}^{m} c_i t^{i-1}$$

that interpolates the data.

Applying the interpolation conditions,

- $P_{m-1}(t_1) = y_1$
- $P_{m-1}(t_2) = y_2$
- •
- $P_{m-1}(t_m) = y_m$

We have systems of equations

$$c_1 + c_2t_1 + c_3t_1^2 + \dots + c_mt_1^{m-1} = y_1$$

$$c_1 + c_2t_2 + c_3t_2^2 + \dots + c_mt_2^{m-1} = y_2$$

$$\vdots$$

$$c_1 + c_2t_m + c_3t_m^2 + \dots + c_mt_m^{m-1} = y_m$$

That is, a linear system

$$\begin{pmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{m-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_m & t_m^2 & \cdots & t_m^{m-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

The matrix V is called the Vandermonde matrix.

#### Lemma 5.2.1

If all the  $t_i$  are distinct, then the Vandermonde matrix is invertible.

Therefore, we can determine the poly coefficients c.

Using IEEE double precision, solving  $V\underline{c} = \underline{y}$  get  $\underline{c}'s$  to approximately 4 significant digits of accuracy. It is difficult to determine interpolants accurately this way.

Originally, we are trying to find  $P_{m-1}(t) \in \mathbb{P}_{m-1}$ , write

$$P_{m-1}(t) = \sum_{i=1}^{m} c_i t^{i-1}$$

using monomial basis

$$\{1, t, t^2, \dots, t^{m-1}\}$$

### 5.2.2 Lagrange Basis

#### Using a Different Basis

Suppose  $\{b_i(t)\}_{i=1}^m$  is a basis for  $\mathbb{P}_{m-1}$ , then

$$P_{m-1}(t) = \sum_{i=1}^{m} \alpha_i b_i(t)$$

for some  $\alpha_i \in \mathbb{R}$  to be determined.

We know that

$$P_{m-1}(t_i) = y_i$$

and using the new basis, we have

$$\alpha_1 b_1(t_1) + \alpha_2 b_2(t_1) + \dots + \alpha_m b_m(t_1) = y_1$$

$$\alpha_1 b_1(t_2) + \alpha_2 b_2(t_2) + \dots + \alpha_m b_m(t_2) = y_2$$

$$\vdots$$

$$\alpha_1 b_1(t_m) + \alpha_2 b_2(t_m) + \dots + \alpha_m b_m(t_m) = y_m$$

which yields the system of equations

$$\begin{pmatrix} b_1(t_1) & b_2(t_1) & \cdots & b_m(t_1) \\ b_1(t_2) & b_2(t_2) & \cdots & b_m(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ b_1(t_m) & b_2(t_m) & \cdots & b_m(t_m) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

#### Remark

The choice of basis is important. We want to choose a basis that is easy to work with.

What if B = I? Then we  $\kappa_I = 1$  and  $\underline{\alpha} = y$ .

We choose a basis such that

$$b_i(t_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

**Example.** Consider R = 1, we want

- $b_1(t_2) = 0$
- $b_1(t_3) = 0$
- :
- $b_1(t_m) = 0$

So if we were to construct a polynomial satisfying these conditions, we would include the factors

$$(t-t_2)$$
  $(t-t_3)$   $\cdots$   $(t-t_m)$ 

which suggests  $b_1(t)$  takes the form

$$(t-t_2)(t-t_3)\cdots(t-t_m)$$

Note that this is of degree m-1, and since our basis cannot have degree more than m-1, we cannot add more factors.

Now we need to satisfy  $b_1(t_1) = 1$ , so we need to divide by a constant. We can choose

$$b_1(t) = \frac{(t-t_2)(t-t_3)\cdots(t-t_m)}{(t_1-t_2)(t_1-t_3)\cdots(t_1-t_m)}$$

 $\Diamond$ 

In general,

$$b_k(t) = \frac{\prod_{j=1, j \neq k}^{m} (t - t_j)}{\prod_{j=1, j \neq k}^{m} (t_k - t_j)}$$

#### Lemma 5.2.2

The set of functions

$$\{b_k(t)\}_{k=1}^m$$

are linearly independent and form a basis for  $\mathbb{P}_{m-1}$ .

This basis is know as the Lagrange basis.

#### **Definition 5.2.3** Lagrange Basis

The Lagrange basis is a set of functions

$$\{l_k(t)\}_{k=1}^m$$
 where  $l_k(t) = \prod_{j=1, j \neq k}^m \frac{t - t_j}{t_k - t_j}$ 

#### Remark

The Lagrange basis has the property

$$l_k(t_j) = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases}$$

The Lagrange form of the interpolating polynomial is

$$P_{m-1}(t) = \sum_{k=1}^{m} c_k l_k(t)$$
 where  $\underline{c} = \underline{y}$ 

**Example.** We use the Lagrange basis to compute  $P_2(t)$  that interpolates

$$\{(-1,2),(1,1),(2,1)\}$$

We have the expressions for the Lagrange basis

$$l_1(t) = \prod_{k=2,3} \frac{t - t_k}{t_1 - t_k} \qquad l_2(t) = \prod_{k=1,3} \frac{t - t_k}{t_2 - t_k} \qquad l_3(t) = \prod_{k=1,2} \frac{t - t_k}{t_3 - t_k}$$

$$= \frac{t - 1}{-1 - 1} \cdot \frac{t - 2}{-1 - 2} \qquad = \frac{t + 1}{1 + 1} \cdot \frac{t - 2}{1 - 2} \qquad = \frac{t + 1}{2 + 1} \cdot \frac{t - 1}{2 - 1}$$

$$= \frac{1}{6}(t - 1)(t - 2) \qquad = -\frac{1}{2}(t + 1)(t - 2) \qquad = \frac{1}{3}(t + 1)(t - 1)$$

 $\Diamond$ 

so

$$P_2(t) = 2l_1(t) + 1l_2(t) + 1l_3(t)$$

$$= 2 \cdot \frac{1}{6}(t-1)(t-2) - \frac{1}{2}(t+1)(t-2) + \frac{1}{3}(t+1)(t-1)$$

$$= \frac{4}{3} - \frac{1}{2}t + \frac{1}{6}t^2$$

which is the same as the polynomial we found earlier.

Remark Lagrange Basis are Expensive

For each  $l_i(t)$ , we require approximately 4m FLOPs.

- For the numerator, m-1 multiplications and m-1 subtractions
- For the denominator, m-1 multiplications and m-1 subtractions
- A finial division

so we require a total of

$$\Theta(m^2)$$
 FLOPs

in total to compute the Lagrange basis.

#### 5.2.3 Newton Basis

Recall for a general basis  $\{b_i(t)\}_{i=1}^m$ , we have

$$P_{m-1}(t) = \sum_{i=1}^{m} \alpha_i b_i(t)$$

and the interpolating conditions

$$P_{m-1}(t_i) = y_i$$

We solve the system of equations

$$Bc = y$$

for  $\underline{c}$ , where

$$B = \begin{pmatrix} b_1(t_1) & b_2(t_1) & \cdots & b_m(t_1) \\ b_1(t_2) & b_2(t_2) & \cdots & b_m(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ b_1(t_m) & b_2(t_m) & \cdots & b_m(t_m) \end{pmatrix}$$

We want a B that is easy to solve, but perhaps not as easy as the identity matrix. We explore a triangular system.

#### Lower Triangular System

We choose a basis  $\{b_i(t)\}_{i=1}^m$  such that

$$B = \begin{pmatrix} b_1(t_1) & 0 & 0 & \cdots & 0 \\ b_1(t_2) & b_2(t_2) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ b_1(t_{m-1}) & b_2(t_{m-1}) & \cdots & b_{m-1}(t_{m-1}) & 0 \\ b_1(t_m) & b_2(t_m) & \cdots & b_{m-1}(t_m) & b_m(t_m) \end{pmatrix}$$

For simplicity, take  $b_1(t) = 1$ .

- Row 1:  $b_2(t_1) = 0, b_3(t_1) = 0, \dots, b_m(t_1) = 0$ We include  $(t - t_1)$  as a factor of each  $b_R(t)$  for  $R = 2, 3, \dots, m$ .
- Row 2:  $b_3(t_2) = 0, b_4(t_2) = 0, \dots, b_m(t_2) = 0$ We include  $(t - t_2)$  as a factor of each  $b_R(t)$  for  $R = 3, 4, \dots, m$ .
- :
- Row m-1:  $b_m(t_{m-1}) = 0$ We include  $(t-t_{m-1})$  as a factor of  $b_m(t)$ .

which suggests

$$b_k(t) = \prod_{j=1}^{k-1} (t - t_j)$$

#### Remark

This is a basis because we have m linearly independent functions.

#### **Definition 5.2.4** Newton Basis

The **Newton basis** is a set of functions

$${n_k(t)}_{k=1}^m$$
 where  $n_k(t) = \prod_{j=1}^{k-1} (t - t_j)$ 

The Newton form of the interpolating polynomial is

$$P_{m-1}(t) = \sum_{k=1}^{m} c_k \prod_{j=1}^{k-1} (t - t_j)$$

#### Remark Newton Basis are Cheaper

If we can remember the results  $t-t_i$ , then each new term added would only require 3 FLOPs. This is a significant improvement over the Lagrange basis. The total operation count is  $\Theta(3m)$  FLOPs

Now, to determine the coefficients, we solve a lower triangular system  $B\underline{c} = y$  We know that

• 
$$y_1 = P_{m-1}(t_1) = c_1$$
 so

$$c_1 = y_1$$

• 
$$y_2 = P_{m-1}(t_2) = c_1 + c_2(t_2 - t_1)$$
 so

$$c_2 = \frac{y_2 - c_1}{t_2 - t_1}$$

:

• 
$$y_m = P_{m-1}(t_m) = \sum_{k=1}^m c_k \prod_{j=1}^{k-1} (t_m - t_j)$$
 so

$$c_m = \frac{y_m - \sum_{j=1}^{m-1} c_j \prod_{k=1}^{j-1} (t_m - t_k)}{\prod_{k=1}^{m-1} (t_m - t_k)}$$

 $\Diamond$ 

**Example.** Determine the Newton form of the degree 2 polynomial that interpolates

$$\{(-1,2),(1,1),(2,1)\}$$

We have the expressions for the Newton basis

- $c_1 = y_1 = 2$
- $c_2 = \frac{y_2 c_1}{t_2 t_1}$ =  $\frac{1 - 2}{1 - (-1)}$ =  $-\frac{1}{2}$

• 
$$c_3 = \frac{y_3 - [c_1 + c_2(t_3 - t_1)]}{(t_3 - t_1)(t_3 - t_2)}$$
  
=  $\frac{1 - [2 - \frac{1}{2}(2 - (-1))]}{(2 - (-1))(2 - 1)}$   
=  $\frac{1}{6}$ 

So the Newton form of the interpolating polynomial is

$$P_2(t) = 2 - \frac{1}{2}(t+1) + \frac{1}{6}(t+1)(t-1) = \frac{4}{3} - \frac{1}{2}t + \frac{1}{6}t^2$$

This is the same polynomial we found earlier with the monomial and Lagrange basis.

## **5.2.4** Summary

	Monomial Basis	Lagrange Basis	Newton Basis
Evaluate $P_{m-1}(t)$	Easy	Difficult	Easy
Integrate/Differentiate	Easy	Difficult	Medium
Finding Coefficients	Can be Ill-conditioned	Very Easy	Easy
$\mathrm{Add}\ (t_{m+1},g_{m+1})$	All Coefficients Change	Just add $\frac{t-t_{m+1}}{t-t_m}$ to each $l_k(t)$	Add $c_m \prod_{j=1}^{m-1} (t - t_j)$ to previous interpolants

What about accuracy? We know

$$P_{m-1}(t_i) = y_i$$
 for all  $i = 1, 2, ..., m$ 

but how can we determine if  $P_{m-1}(t)$  is a good approximation for f(t) for  $t \neq t_i$ ?

#### Theorem 5.2.5

Suppose the polynomial  $P_{m-1}(t)$  interpolates the function f(t) at distinct points  $t_1 < t_2 < \cdots < t_m$ . If f is m-times differentiable on the interval  $[t_1, t_m]$ , then for any point  $t \in [t_1, t_m]$ ,

$$f(t) - p_{m-1}(t) = \frac{1}{m!} f^{(m)}(c_t) \omega_m(t)$$

where

•  $c_t$  is some point in  $t_1, t$ , that depends on t;

• 
$$\omega_m(t) = (t - t_1)(t - t_2) \cdots (t - t_m) = \prod_{j=1}^m (t - t_j)$$

We have

$$\max_{t \in [t_1, t_m]} |f(t) - P_{m-1}(t)| \le \frac{1}{m!} \max_{t \in [t_1, t_m]} |f^{(m)}(t)| \max_{t \in [t_1, t_m]} |\omega_m(t)|$$

- $\bullet$  m is the interpolation points we can control
- $f^{(m)}(t)$  is the m-th derivative of f(t) which is the function we are trying to approximate
- depends on placement of  $t_i$  in the interval  $[t_1, t_m]$
- We can show that

$$\max_{t \in [t_1, t_m]} |\omega_m(t)| \le \frac{h^m \cdot (m-1)!}{4}$$

where

$$h = \max_{i=1,\dots,m-1} |t_{i+1} - t_i|$$

Thus,

$$\max_{t \in [t_1, t_m]} |f(t) - P_{m-1}(t)| \le \frac{1}{m!} \max_{t \in [t_1, t_m]} |f^{(m)}(t)| \frac{h^m \cdot (m-1)!}{4}$$
$$\le \frac{M \cdot h^m}{4m}$$

where

$$M = \max_{t \in [t_1, t_m]} |f^{(m)}(t)|$$

PART II

APPENDICES

# BIBLIOGRAPHY

- [1] Danelnov, Plantilla latex, https://github.com/Danelnov/Plantilla-latex, 2022.
- [2] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. DOI: 10.1109/IEEESTD.2019.8766229 (cited on page 8).
- [3] Material Design. "The color system." Section: Tools for Picking Colors. (2024), [Online]. Available: https://m2.material.io/design/color/the-color-system.html#tools-for-picking-colors.

# INDEX

```
Catastrophic Cancellation, 15
Condition Number, 19
Cramer's Rule, 23
Floating Point
    Base, 7
    Exponent, 7
    Mantissa, 7
    Significant, 7
Floating Point Representation, 7
FLOP, 24
Frobenius Norm, 39
Ill-Conditioned Problem, 20
Lagrange Basis, 50
Machine Epsilon, 12
Newton Basis, 52
Non-Singular Matrix, 23
Normwise relative error, 37
Normwise relative residual, 37
Numerical Stability, 7
Numerically Stable Algorithm, 17
Numerically Unstable, 6
Pivot Position, 42
Relative Round-off Error Bound, 12
Residual, 36
Signed Relative Error, 19
Well-Conditioned Problem, 20
```