CS 371

Introduction to Computational Mathematics

Jiucheng Zang

January 5th, 2026

Notes compiled for CS 371.

Contents

CHAPTER 1

Introduction

- Symbolic: $\sqrt{2}$
  - correctness
  - efficiency
- Numeric: 1.41421356
  - correctness
  - efficiency
  - stability
  - conditioning

Office: MC 6431

Desire of Algorithms:

- Accuracy: produce result that is numerical very close to true value
- Efficiency: quickly solve the problem with reasonable computational resources
- Robustness: algorithm should work well for variety of inputs

Problem Conditioning:

- Maybe sensitive to small changes in input
- Can we get a good solution numerically in such cases?

Algorithm Stability:

- Some may work for all inputs, but produce large numerical errors for some inputs
- Some will only work for some inputs, but produce small numerical errors for those inputs
- How/Why to choose one over the other?

CHAPTER 2

Errors and Error Propagation

2.1 Source of Error:

- Error in input
  - measurement error: $\Delta x = |x - \tilde{x}|$, x is true value, $\tilde{x}$ is measured value
    e.g., $x^2 - 0.999$ may given as $x^2 - 1$
  - rounding error: Cause by a difference between the exact value x and the computational or floating point representation $\hat{x} = fl(x)$
    e.g. Let $x = 0.003456978$, as input for a 4-digit floating point system, we have $\hat{x} = fl(x) = 0.3457 \times 10^{-2}$
    Error: $|x - \hat{x}| = 0.000000022$
- Error as a result of algorithm
  - truncation error
    e.g. $\sum_{i=1}^{\infty} a_i x^i$ approximated by $\sum_{i=1}^{n} a_i x^i$
    Error: $|\sum_{i=n+1}^{\infty} a_i x^i|$
  - rounding error in elementary step of algorithm

Example

Is calculus $(\sqrt{3})^2 = 3$. Do we have precisely $(\sqrt{3})^2 = 3$ with computer arithmetic too?

No, not in general. In real-number arithmetic, $(\sqrt{3})^2 = 3$ exactly. However, in computer arithmetic, the $\sqrt{3}$ cannot be represented exactly, because it has an infinite binary expansion. When this approximation is squared, the result is very close to 3, but it generally not exactly equal to 3 due to rounding errors.

Example

Try addition of $a = 2.0126 \times 10^4$ and $b = 5.6271 \times 10^5$ in a 4-digit-precision computer.

Normalize and round:
$a = 0.20126 \times 10^5 \implies 0.2013 \times 10^5$ $b = 0.56271 \times 10^6 \implies 0.5627 \times 10^6$
Alisgn exponents: $0.2013 \times 10^5 \implies 0.02013 \times 10^6 \implies 0.0201 \times 10^6$
Addition: $(0.0201 + 0.5627) \times 10^6 = 0.5828 \times 10^6$

Definition 2.1.

Absolute error: $|\tilde{x} - x|$, where $\tilde{x}$ is an approximation of x
Relative error: $\frac{|\tilde{x}-x|}{|x|}$, assuming $x \neq 0$

Example

Determine the absolute and relative error in the following cases:

- $p = 0.30012 \times 10^1, p^* = 0.30200 \times 10^1$

$$E_{abs} = |p^* - p| = |0.30200 \times 10^1 - 0.30012 \times 10^1| = 0.0188$$
$$E_{rel} = \frac{|p^* - p|}{|p|} = \frac{0.00188 \times 10^1}{0.30012 \times 10^1} \approx 6.27 \times 10^{-3}$$

- $p = 0.30012 \times 10^{-2}, p^* = 0.30200 \times 10^{-2}$

$$E_{abs} = |p^* - p| = |0.30200 \times 10^{-2} - 0.30012 \times 10^{-2}| = 1.88 \times 10^{-5}$$
$$E_{rel} = \frac{|p^* - p|}{|p|} = \frac{0.00188 \times 10^{-2}}{0.30012 \times 10^{-2}} \approx 6.27 \times 10^{-3}$$

Conclusion: Absolute error alone can be misleading when comparing quantities of different scales. Relative error gives a meaningful measure of accuracy independent of units or magnitude.

Example

Evaluate $f(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ at $x = 4.71$ using three-digit arithmetic. Can you introduce an alternative approach to decrease the round-off error?

Direct evaluation:

$$\begin{aligned}
x^2 &= 4.71 \times 4.71 = 22.1841 \approx 22.2 \\
x^3 &= 22.2 \times 4.71 = 104.562 \approx 105 \\
6.1x^2 &= 6.1 \times 22.2 = 135.42 \approx 135 \\
3.2x &= 3.2 \times 4.71 = 15.072 \approx 15.1 \\
f(4.71) &= 105 - 135 + 15.1 + 1.5 = -13.4 \\
\text{exact } f(4.71) &\approx -14.263899 \\
E_{rel} &= \frac{|-14.263899 + 13.4|}{|-14.263899|} \approx 0.06
\end{aligned}$$

Alternative approach: Horner's method

$$\begin{aligned}
f(x) &= ((x - 6.1)x + 3.2)x + 1.5 \\
x - 6.1 &= 4.71 - 6.1 = -1.39 \\
(-1.39) \times 4.71 &= -6.5469 \approx -6.55 \\
-6.55 + 3.2 &= -3.35 \\
(-3.35) \times 4.71 &= -15.7785 \approx -15.8 \\
-15.8 + 1.5 &= -14.3 \\
E_{rel} &= \frac{|-14.263899 + 14.3|}{|-14.263899|} \approx 0.0025
\end{aligned}$$

Truncation Error: Taylor series

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number a is the power series.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Consider the finite sum (the n-th degree Taylor polynomial of f centered at a):

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

and this is called the **nth Taylor polynomial**

The truncation error is defined as

$$R_n(x) = \frac{f^{(n+1)}(\xi_n(x))}{(n+1)!}(x-a)^{n+1}$$

and is called the **Truncation Error** associated with $p_n(x)$, where $\xi_n(x)$ is some number between a and x.

$$f(x) = P_n(x) + R_n(x)$$

Example

What is the largest error which might result from using the first three terms of the series to approximate $f(x) = \cos(x)$ at $a = 0$, if $x \in [-\pi, \pi]$.

$$R_n(x) = \frac{f^{(n+1)}(\xi_n(x))}{(n+1)!}(x-a)^{n+1}$$

For $n = 3$, we have

$$R_3(x) = \frac{f^{(4)}(\xi_3(x))}{4!}(x-0)^4 = \frac{\cos(\xi_3(x))}{4!}x^4 \quad (1)$$

$$\leq \frac{1}{4!}(\pi)^4 \quad (2)$$

Example

Find the smallest value of n for which the n^{th} degree Taylor series for $f(x) = e^{2x}$ at $a = 0$ approximates e^{2x} on the interval $0 \leq x \leq 1$ with an error no greater than 10^{-6}.

$$|R_n(x)| = \left|\frac{f^{(n+1)}(\xi_n(x))}{(n+1)!}(x-a)^{n+1}\right| = \left|\frac{2^{n+1}e^{2\xi_n(x)}}{(n+1)!}x^{n+1}\right|$$

For $0 \leq x \leq 1$, the maximum value of $e^{2\xi_n(x)}$ is e^2 and the maximum value of x^{n+1} is 1. Thus,

$$|R_n(x)| \leq \frac{2^{n+1}e^2}{(n+1)!} \leq 10^{-6}$$

Using trial and error, we find that $n = 13$ satisfies the inequality.

Definition 2.2. Catastrophic Cancellation

The cancellation of the significant digits makes the unknown rounded-off digits relevant to the final result. We get a lousy final approximation because we don't know the lost digits. For this reason the cancellation of significant digits is called Catastrophic Cancellation.

Solution: Using the another approach (rewriting the function to avoid subtraction of)

All the terms in the alternative method are positive, so there can be no cancellation of information.

Example

Applying Taylor series to $f(x) = e^x$ at $a = 0$, gives

$$e^x = e^0 + \frac{e^0}{1!}x + \frac{e^0}{2!}x^2 + \cdots = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Plugging in $x = -5.5$, and using 5 digits of accuracy, we have

$$e^{-5.5} \approx 0.0026363$$

The actual value is $e^{-5.5} \approx 0.0040868$, there is a huge error.

Another approximation is given by

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

With 5 digits of accuracy, we now get $e^{-5.5} \approx 0.0040865$, which is much more accurate.

2.2 Floating Point Numbers and Operations

Represent real numbers on a computer

- Infinite in extent: there exists x such that $|x|$ is arbitrarily large.
- Infinite in density: Any interval (a, b) contains infinitely many numbers.

On a computer, only a finite number of digits can be stored before and after the decimal

Solution: Choices **fixed point** and **floating point** representation

- A fixed point number system is characterized by 3 values
  - b, base
  - I, number of digits for integer part
  - F, number of digits for fractional part
- Numbers are of the form $\pm i_1 i_2 \ldots i_I . f_1 f_2 \ldots f_F$ where $i_k, f_k \in \{0, 1, \ldots, b-1\}$

Definition 2.3. Floating point representation

- Base: base of the number system, b_f.
- The mantissa: contains the normalized value of the number, m_f.
- Exponent: which defines the offset from normalization, e_f.

$F[b = b_f, m = m_f, e = e_f] \equiv \pm 0.x_1 x_2 \ldots x_m \times b^{\pm y_1 y_2 \ldots y_{e_f}}$,

where $1 \leq x_1 \leq b-1$ and $0 \leq x_i \leq b-1$ for $i = 2, 3, \ldots, m$.

Compare fixed point and floating point representation

Fixed Point	**Floating Point**
Values are evenly spaced	Values are not evenly spaced, smaller values are closer together
Really small or really large values cannot be represented	Greater range of values can be represented (large and small)
To represent a real number, choose the "closest" computer value (round or chop)	Again, rounding or chopping is required when choosing a representation of a value

Example

Consider a fictitious computer with a floating point number system with parameters $b = 3$, $m = 4$, and $e = 2$. i.e. $F[3, 4, 2]$. Represent $x = 0.0011220212$ in this system.

$$\hat{x} = 0.1122021 \times 3^{-2}$$

Since $m = 4$: $fl(x) = \hat{x} = 0.1122 \times 3^{-2}$

Example

Consider a binary computer with a floating point number system $F[b = 2, m = 5, e = 3]$, represent $x = 11010.101$ in this system.

$\hat{x} = 0.11010101 \times 2^5$

$m = 5$, $fl(x) = \hat{x} = 0.11010 \times 2^5, (5)_2 = (101)_2$

$fl(x) = \hat{x} = 0.11010 \times 2^{101}$

2.3 Standard floating point system

Definition 2.4. **Single Precision:** This is one of the standard floating point number system, where numbers are represented on a computer in a 32-bit memory (4 bytes).

$$s_m, b_1 b_2 \dots b_{\mathbf{23}}, s_e, e_1 e_2 \dots e_{\mathbf{7}}$$

where s_m and s_e are sign bits (**0 for positive, 1 for negative**).

$$(-1)^{s_m} \times (0.b_1 b_2 \dots b_{23}) \times 2^{\pm \text{exponent}}, b_1 = 1$$

Example

What's the largest number that can be represented in single precision?

$$\underbrace{(-1)^0}_{\text{positive}} \mid \underbrace{11\dots1}_{\text{23 bits(mantissa)}} \mid \underbrace{(-1)^0}_{\text{positive}} \mid \underbrace{11\dots1}_{\text{7 bits(exponent)}}$$

Mantissa part:

$$\begin{aligned}(0.\underbrace{11\ldots1}_{\text{23 bits}})_2 &= (1\times 2^{-1}) + (1\times 2^{-2}) + \cdots + (1\times 2^{-23})\\ &= \frac{1}{2} + (\frac{1}{2})^2 + \cdots + (\frac{1}{2})^{23}\\ &= \frac{1}{2}\left(1 + \frac{1}{2} + \cdots + (\frac{1}{2})^{22}\right)\\ &= \frac{1}{2}\left(\sum_{i=0}^{22}(\frac{1}{2})^i\right)\\ &= \frac{1}{2}\left(\frac{1-(\frac{1}{2})^{23}}{1-\frac{1}{2}}\right)\\ &= 1 - 2^{-23}\end{aligned}$$

Exponent part:

$$(\underbrace{11\ldots1}_{\text{7 bits}})_2 = 2^7 - 1 = 127$$

Therefore, the largest number in single precision is:

$$\begin{aligned}&= 1\times\left(1-2^{-23}\right)\times 2^{127}\\ &\approx 2^{127}\\ &\approx 1.7\times 10^{38}\end{aligned}$$

Example

What's the smallest positive number that can be represented in single precision?

$$\underbrace{(-1)^0}_{\text{positive}} \mid \underbrace{00\ldots1}_{\text{23 bits(mantissa)}} \mid \underbrace{(-1)^1}_{\text{negative}} \mid \underbrace{11\ldots1}_{\text{7 bits(exponent)}}$$

Mantissa part:

$$(0.\underbrace{00\ldots1}_{\text{23 bits}})_2 = 2^{-23}$$

Exponent part:

$$-(\underbrace{11\ldots1}_{\text{7 bits}})_2 = -(2^7 - 1) = -127$$

Therefore, the smallest positive number in single precision is:

$$\begin{aligned} &= 1 \times 2^{-23} \times 2^{-127} \\ &= 2^{-150} \\ &\approx 7.0 \times 10^{-46} \end{aligned}$$

Definition 2.5. **Double Precision:**

$$s_m, b_1 \dots b_{\mathbf{52}}, s_e, e_1 \dots e_{\mathbf{10}}$$

Note:

- The largest number in double precision is approximately 9×10^{307}
- The smallest positive number in double precision is approximately 2.5×10^{-324}

Example

Consider the floating point number system $F[10, 3, 2]$, corresponding to base 10. Non-zero numbers have the form $\pm 0.d_1d_2d_3 \times 10^{\pm e_1 e_2}$, where $d_1 \neq 0$.

Q: What is the largest and smallest positive numbers that can be stored in F.

The largest positive number: 0.999×10^{99}
The smallest positive number: 0.100×10^{-99}

Q: How many non-zero numbers(positive and negative) can be stored in F.

d_1 has 9 possible choices (1-9)
d_2, d_3 have 10 possible choices (0-9)
In total for mantissa: $9 \times 10^2 = 900$

In total for exponent: $99 + 99 + 1 = 199$ (including 0)

Therefore, total non-zero numbers in F is: $2 \times 900 \times 199 = 358200$

Definition 2.6. **Decimal Machine Numbers**

Normalized decimal floating-point form is:

$$\pm 0.d_1 d_2 \dots d_k \times 10^n, 1 \leq d_1 \leq 9, 0 \leq d_i \leq 9, i = 2, 3, \dots, k$$

Any positive real number can be normalized to the form:

$$y = 0.d_1 d_2 \dots d_k d_{k+1} \cdots \times 10^n$$

The floating point form of y, $fl(y)$, is obtained by terminating the mantissa of y at k decimal digits

There are two ways to termination:

- chopping: just simply chop off the digits $d_{k+1} d_{k+2} \dots$
- rounding: when $d_{k+1} \geq 5$, increase d_k by 1; otherwise, just chop off the digits

Definition 2.7.

The **machine epsilon**, e_{mach}, is the smallest number $\epsilon > 0$ such that $fl(1 + \epsilon) > 1$.

Proposition

The machine epsilon is given by:

1. $e_{mach} = b^{1-m}$ if chopping is used
2. $e_{mach} = \frac{1}{2} b^{1-m}$ if rounding is used

Proof

Consider a normalized floating point system with chopping

$$1 = 0.\underbrace{100 \dots 0}_{m \text{ digits}} \times b^1$$

$$1 + \epsilon = 0.\underbrace{100 \dots 1}_{m \text{ digits}} \times b^1$$

Subtracting the two, we get:

$$\begin{aligned}\epsilon &= (0.100\ldots 1 - 0.100\ldots 0) \times b^1 \\ &= (0.000\ldots 1) \times b^1 \\ &= b^{1-m}\end{aligned}$$

□

Theorem 2.1

For any floating point system F, under chopping

$$|\delta_x| = \left|\frac{x - fl(x)}{x}\right| \leq \epsilon_{mach}$$

Therefore,

- under single precision: $|\delta_x| \leq 0.24 \times 10^{-6}$
- under double precision: $|\delta_x| \leq 0.44 \times 10^{-15}$

Note: Since $\delta_x = \frac{x - fl(x)}{x}$, we may also write $fl(x) = x(1 + \delta_x)$, with $|\delta_x| \leq \epsilon_{mach}$.

Hence, we often write:

$$fl(x) = x(1 + \eta), |\eta| \leq \epsilon_{mach}$$

Proof

$x = 0.d_1 d_2 \ldots d_m d_{m+1} d_{m+2} \ldots$

Under chopping, $fl(x) = 0.d_1 d_2 \ldots d_m$

$$\begin{aligned}\frac{x - fl(x)}{x} &= \frac{0.00\ldots 0d_{m+1}d_{m+2}\ldots}{0.d_1d_2\ldots d_md_{m+1}d_{m+2}\ldots}\\ &\leq \frac{0.00\ldots 0d_{m+1}d_{m+2}\ldots}{0.1}\\ &< \frac{b^{-m}}{1}\\ &= b^{1-m}\\ &= \epsilon_{mach}\end{aligned}$$

□

Definition 2.8. The symbol $\oplus$ is used in define floating point addition, defined as:

$$a \oplus b = fl(fl(a) + fl(b))$$

Proposition

For any floating point number system F,

$$a \oplus b = fl(fl(a) + fl(b)) = (fl(a) + fl(b))(1+\eta), |\eta| \leq e_{mach}$$

This may also be written as:

$$a \oplus b = (a(1+\delta_a) + b(1+\delta_b))(1+\eta), |\delta_a|, |\delta_b|, |\eta| \leq e_{mach}$$

In general the operation of addition under F is not associative, i.e.

$$(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$$

Example

Suppose that $x = \frac{5}{7}$ and $y = \frac{1}{3}$. Use five-digit chopping to compute $x + y$.

$$\begin{aligned}\hat{x} &= fl(x) = 0.71428 \times 10^0 \\ \hat{y} &= fl(y) = 0.33333 \times 10^0 \\ \hat{z} &= fl(\hat{x} + \hat{y}) = fl(0.71428 + 0.33333) \\ &= fl(1.04761) = 0.10476 \times 10^1\end{aligned}$$

True value: $x + y = \frac{22}{21}$

$$E_{rel} = \frac{\left|\frac{22}{21} - 1.0476\right|}{\frac{22}{21}} \approx 1.85 \times 10^{-5}$$

2.4 Condition Number

Definition 2.9.

- A problem is **well-conditioned** with respect to the absolute error if small changes in the input, result in small changes in the output.
- A problem is **ill-conditioned** with respect to the absolute error if small changes in the input, result in large changes in the output.

Input I, $I + \Delta I$, where $|\Delta I|$ is a small change output O, $O + \Delta O$, for some ΔO.

Definition 2.10. Condition Number

- Condition number with respect to the absolute error is defined as: $\kappa_A = \frac{||\Delta z||}{||\Delta x||}$, and it is called the **absolute condition number**, where Δx is change in the input and Δz is change in the output.
- Condition number with respect to the relative error is defined as:

$$\kappa_R = \frac{||\Delta z||/||z||}{||\Delta x||/||x||} = \frac{||\Delta z|| \cdot ||x||}{||z|| \cdot ||\Delta x||}$$

  and it called the **relative condition number**.
- For $\underline{0.1 \leq \kappa_A, \kappa_R < 10}$, problem is **well-conditioned** and for $\underline{\kappa_A, \kappa_R \to \infty}$ problem is **ill-conditioned**.

2.5 Vector Norm

Definition 2.11. Suppose V is a vector space over $\mathbb{R}^n$, Then $||\cdot||$ is a vector norm on V if and only if $||\vec{v}|| \geq 0$, and

- $||\vec{v}|| = 0$ if and only if $\vec{v} = \vec{0}$
- $||\alpha\vec{v}|| = |\alpha| \cdot ||\vec{v}||$ for any scalar $\alpha \in \mathbb{R}$, and vector $\vec{v} \in V$
- $||\vec{v} + \vec{w}|| \leq ||\vec{v}|| + ||\vec{w}||$ for any vectors $\vec{v}, \vec{w} \in V$ (Triangle Inequality)

Definition 2.12. The **2-norm** over $\mathbb{R}^n$ is defined as:

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

Definition 2.13. The **∞-norm** over $\mathbb{R}^n$ is defined as:

$$||\vec{x}||_\infty = \max_{1\leq i\leq n} |x_i|$$

Definition 2.14. The **1-norm** over $\mathbb{R}^n$ is defined as:

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

Theorem 2.2

Cauchy-Schwarz Inequality:

For any vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$,

$$|\vec{u} \cdot \vec{v}| \leq ||\vec{u}|| \cdot ||\vec{v}||$$

Example

Consider a problem $y = \frac{x}{1-x}$. Is this a well conditioned problem?

Case 1: x close to 1

Let $x = 0.93$, then $y = \frac{0.93}{1-0.93} = 13.285714$ perturb the input slightly: $\tilde{x} = 0.94$, So $\Delta x = 0.01$

Relative error $= \frac{|\Delta x|}{|x|} = \frac{0.01}{0.93} \approx 0.01075$ New output: $\tilde{y} = \frac{0.94}{1-0.94} = 15.66667$ Relative error $= \frac{|\Delta y|}{|y|} = \frac{2.380956}{13.285714} \approx 0.1792$

Condition number:

$$\kappa_R = \frac{0.1792}{0.01075} \approx 16.67$$

Thus, a small change in input results in a large change in output, so the problem is **ill-conditioned**.

Case 2: x away from 1

Let $x = 5$, and perturb it to $\tilde{x} = 5.06$, so $\Delta x = 0.06$

$y = \frac{5}{1-5} = -1.25$, and $\tilde{y} = \frac{5.06}{1-5.06} \approx -1.246305$

Relative error in input: $\frac{0.06}{5} = 0.012$ Relative error in output: $\frac{|-1.246305+1.25|}{|-1.25|} \approx 0.003$

Condition number:

$$\kappa_R = \frac{0.003}{0.012} = 0.25$$

Thus, a small change in input results in a small change in output, so the problem is **well-conditioned**.

Note

Conditioning is a property of the problem, not of the algorithm or its implemntation.

2.6 Stability of algorithms

Definition 2.15.

If any initial error in the data is magnified by an algorithm, the algorithm is considered numerically unstable.

Can lead to meanings results, for **seemingly** reasonable methods on reasonable problems.

Small changes in the initial data, produce small changes in the final result. An algorithm that satisfy this property is called **Stable**.

Definition 2.16. **Stability of a Numerical Algorithm**

If $E_0 > 0$ denotes an error introduced at some steps in the calculations and En represents the magnitude of error after n subsequent operations:

- If $E_n \approx C_n E_0$ (C is a constant) then the growth of error is linear.
- If $E_n \approx C^n E_0$ for $C > 1$, then the growth of error is called exponential.

Therefore, algorithm with linear growth of error is stable whereas an algorithm exhibiting exponential error growth is unstable.

Example

Consider the integration problem

$$I_n = \int_0^1 \frac{x^n}{x+\alpha} dx$$

for a given n where α is some fixed constant.

A recursive algorithm to solve it, for $n \geq 0$:

$$I_0 = \log \frac{1+\alpha}{\alpha}, \quad I_n = \frac{1}{n} - \alpha I_{n-1}$$

This is a well-conditioned problem.

Let $(I_n)_E$ be the exact value of I_n.
Let $(I_n)_A$ be the approximated value of I_n.

ϵ_0 is the difference between the exact and approximated value of I_0 and the approximated value of I_0:

$$\epsilon_0 = (I_0)_A - (I_0)_E$$

So, ϵ_0 is the initial error in the data.

Then we want to analyze how ϵ_0 propagates to ϵ_n.

$$\begin{aligned}
\epsilon_n &= (I_n)_A - (I_n)_E \\
&= \left(\frac{1}{n} - \alpha(I_{n-1})_A\right) - \left(\frac{1}{n} - \alpha(I_{n-1})_E\right) \\
&= \left(\frac{1}{n} - \frac{1}{n}\right) - \alpha\left((I_{n-1})_A - (I_{n-1})_E\right) \\
&= -\alpha\epsilon_{n-1} \\
&= -\alpha(-\alpha \cdot \epsilon_2) \\
&\vdots \\
&= (-\alpha)^n \epsilon_0
\end{aligned}$$

If $|\alpha| > 1$, it grows fast, so the algorithm is unstable.
If $|\alpha| < 1$, the error decreases, so the algorithm is stable.

Hence, that recurrence algorithm is unstable for solving

$$I_n = \int_0^1 \frac{x^n}{x+\alpha} dx \quad \text{when } |\alpha| > 1.$$

$I_{100} = 6.64 \times 10^{-3}$ for $\alpha = 0.5$
$I_{100} = 2.1 \times 10^{22}$ for $\alpha = 2.0$

Example

For any constants c_1 and c_2, $p_n = c_1(\frac{1}{3})^n + c_2 3^n$ is a solution to the recursive equation:

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

Investigate the stability of this procedure if $p_0 = 1$ and $p_1 = \frac{1}{3}$.

Given initial values allow us to uniquely determine $c_1 = 1$, $c_2 = 0$ by solving the system of equations:

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

- F is not $\mathbb{R}$
- We can use round-off error analysis to roughly bound the errors incurred by floating point operations.
- We can analyze whether errors accumulate or shrink to determine the stability of algorithms.

CHAPTER 3

Root Finding

Question: Why do we need numerical methods for root finding?

Suppose $N(t)$ denotes the number in the population at time t, and λ denotes the constant birth rate of the population.

Then the population satisfies the differential equation

$$\frac{dN}{dt} = \lambda N(t)$$

with the solution

$$N(t) = N(0)e^{\lambda t}$$

where N_0 is the initial population at time $t = 0$.

Now if immigration is permitted at a constant rate ν, then:

$$\frac{dN}{dt} = \lambda N(t) + \nu$$

with $N(t) = N_0 e^{\lambda t} + \frac{\nu}{\lambda}(e^{\lambda t} - 1)$ as its solution.

After one year:

$$N(1) = N_0 e^{\lambda} + \frac{\nu}{\lambda}(e^{\lambda} - 1)$$

To find λ, we must solve this question for λ. However, it is impossible to solve explicitly for λ in this question.

However, numerical methods can approximate solutions of equations of this type with arbitrary accuracy.

3.1 General Root-Finding Problem

Given any function $f(x)$, find x^* such that $f(x^*) = 0$. x^* is called a root of the equation $f(x) = 0$.

Challenges

- Even if the existence of a root of a function is guaranteed, there is no guaranteed method for finding x^*
- Since x^* may not be defined in our floating point system, we cannot find x^* exactly.

Computational Solution: Given any $f(x)$ and some **error tolerance** $\epsilon > 0$, find x^* such that $|f(x^*)| < \epsilon$.

Definition 3.1. We say x^* is a **double root** of $f(x) = 0$ iff $f(x^*) = 0$ and $f'(x^*) = 0$.

Theorem 3.1 Intermediate Value Theorem

If $f(x)$ is continuous on $[a, b]$ and $y \in [f(a), f(b)]$ then $\exists x^* \in [a, b]$ s.t. $f(x^*) = y$

Note: There could be more than one crossing of y in such an interval.

Note

If we can find $[a, b]$ such that the produce $f(a)f(b) < 0$, then $\exists x^* \in [a, b]$ s.t. $f(x^*) = 0$ since zero is between $f(a)$ and $f(b)$.

3.1.1 Bisection Method

Definition 3.2. Bisection Method

Given $f(x)$ and $[a, b]$, s.t. $f(a)f(b) < 0$, approximate the root with the midpoint $c = \frac{a+b}{2}$. If $f(a)$ and $f(c)$ have opposite signs, then repeat (recurse) on the sub-interval $[a, c]$. Otherwise, recurse on $[c, b]$.

Algorithm 1: Bisection Method

```
Input: $f(x)$, interval $[a,b]$ s.t. $f(a)f(b) < 0$, tolerance $\epsilon > 0$
Output: Approximation of root $x^*$ s.t. $|f(x^*)| < \epsilon$
while $(b-a)/2 > \epsilon$ do
    $c \leftarrow (a+b)/2$;
    if $f(c) = 0$ then
        return $c$
    else if $f(a)f(c) < 0$ then
        $b \leftarrow c$;
    else
        $a \leftarrow c$;
return $(a+b)/2$
```

Note

- This method always works, provided $f(x)$ is continuous.
- The interval length is cut by half at each step.
- After n steps the interval becomes $[a_n, b_n]$ where either $a_n = x_n$ or $b_n = x_n$, implying that $|x_n - x^*| < b_n = a_n$.
- Bisection implies that the interval lengths satisfy:

$$b_n - a_n = \frac{b_0 - a_0}{2^n}$$

Stopping condition

If we stop at $|b_n - a_n| < tol$ for some tolerance value, then

$$\begin{aligned} 2^{-n}(b_0 - a_0) &< tol \\ \implies 2^n &> \frac{|b_0 - a_0|}{tol} \\ \implies n &> \log_2\left(\log\left(\frac{|b_0 - a_0|}{tol}\right)\right) \end{aligned}$$

Example

Show that $f(x) = x^3 + 4x^2 - 10 = 0$ has a root in $[1, 2]$. Use the bisection method to determine an approximation to the root that is accurate to at least within 10^{-1}.

Existence of a root in $[1, 2]$:

$$f(1) = -5, \quad f(2) = 2 \implies f(1)f(2) < 0$$

So, by the Intermediate Value Theorem, there exists a root in $[1, 2]$.

Iteration	Interval $[a_n, b_n]$	Midpoint c_n	$f(c_n)$	Next Interval
1	$[1, 2]$	1.5	$2.375 > 0$	$[1, 1.5]$
2	$[1, 1.5]$	1.25	$-1.796875 < 0$	$[1.25, 1.5]$
3	$[1.25, 1.5]$	1.375	$0.162109 > 0$	$[1.25, 1.375]$
4	$[1.25, 1.375]$	1.3125	$-0.848389 < 0$	$[1.3125, 1.375]$

$$x^* \approx \frac{1.3125 + 1.375}{2} = 1.34375 \quad \text{with error } < 0.1$$

Example

Determine the number of iterations necessary to solve $f(x) = x^3 + 4x^2 - 10 = 0$ with accuracy 10^{-3} using $a = 1, b = 2$.

$$n \geq \frac{1}{\log 2} \log\left(\frac{b-a}{tol}\right) = \frac{1}{\log 2} \log(1000) \approx 10$$

Therefore, I will need $n = 10$ iterations.

3.1.2 Fixed Point Iteration

Definition 3.3. **Fixed Point Iteration**

If $f(x) = 0$, then clearly $x - x + f(x) = 0$, and thus any root of $f(x) = 0$ is also a fixed point of the function $g(x) = x + f(x)$

Example

Determine all fixed points of $g(x) = x^2 - 2$.

A fixed point p of g satisfies $p = g(p) = p^2 - 2$, or equivalently, $p^2 - p - 2 = 0$. Solving this quadratic equation, we find the fixed points are $p = 2$ and $p = -1$.

For $f(x) = x^2 - x - 2$, some possibilities might be:

- $g(x) = x^2 - 2$
- $g(x) = \sqrt{x+2}$
- $g(x) = 1 + \frac{2}{x}$
- $g(x) = x - \frac{x^2-x-2}{m} \quad (m \neq 0)$

One can derive such possibilities starting from $f(x) = 0$, and rearranging to isolate an x by itself on the left side.

3.1.3 Newton's Method

Definition 3.4. Newton's Method

Suppose $f(x)$ is continuous with bounded derivatives $f'(x)$, $f''(x)$, and $f'(x) \neq 0$ then (by Taylor's Theorem):

$$f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + O((x^* - x_k)^2)$$

Ignoring the higher order terms and noting $f(x^*) = 0$,

$$0 \approx f(x_k) + f'(x_k)(x^* - x_k) \implies x^* \approx x_k - \frac{f(x_k)}{f'(x_k)}$$

Note: This does not give us x^* exactly, because we ignored $O((x^* - x_k)^2)$ terms.

Algorithm 2: Newton's Method

```
Input: $f(x)$, initial guess $x_0$, tolerance $\epsilon > 0$
Output: Approximation of root $x^*$ s.t. $|f(x^*)| < \epsilon$
while $|f(x_n)| > \epsilon$ do
    $x_{n+1} \leftarrow x_n - \frac{f(x_n)}{f'(x_n)}$;
    $n \leftarrow n + 1$;
return $x_n$
```

Note

Approximate Derivatives

In practice, $f'(x)$ itself may not be known explicitly (or may be expensive, etc.) For example, if $f(x)$ is a black-box function you can't look inside.

However, approximate it, which will lead to the **Secant Method**.

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

3.1.4 Secant Method

Definition 3.5. Secant Method

Given two initial approximations x_0 and x_1, the secant method generates a sequence according to the formula:

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

Note:

- We need two starting values x_{k-1}, x_k to get x_{k+1}. Use another method (e.g., bisection) to get the first two values.
- Secant converges slower than Newton's method, due to approximating $f'(x_k)$ by finite differences.

3.2 Stopping Criteria

Note

For all of these methods, we need to decide when to stop! Possible options:

- Maximum number of steps taken - prevent infinite loop
- Tolerance on the size of the correction (change): $|x_{k+1} - x_k| < tol$
- Tolerance on the side of the function value $|f(x_k)| < tol$

Stopping Criteria on Bisection

From the diagram above: $|x_{k+1} - x_k| = \frac{b-a}{2}$.

Stopping criteria $\implies \frac{|b_k - a_k|}{2} < tol \implies |x_{k+1} - x_k| < \frac{|b_k - a_k|}{2} < tol$.

For bisection, stopping based on criterion gives a guaranteed error bound.

Notes: This is not true in general for the Newton, Secant, or Fixed-Point methods.

Consider the function $f(x) = \cos(x) - x = 0$. Approximate a root of f using:

- Fixed Point Iteration
- Newton's Method

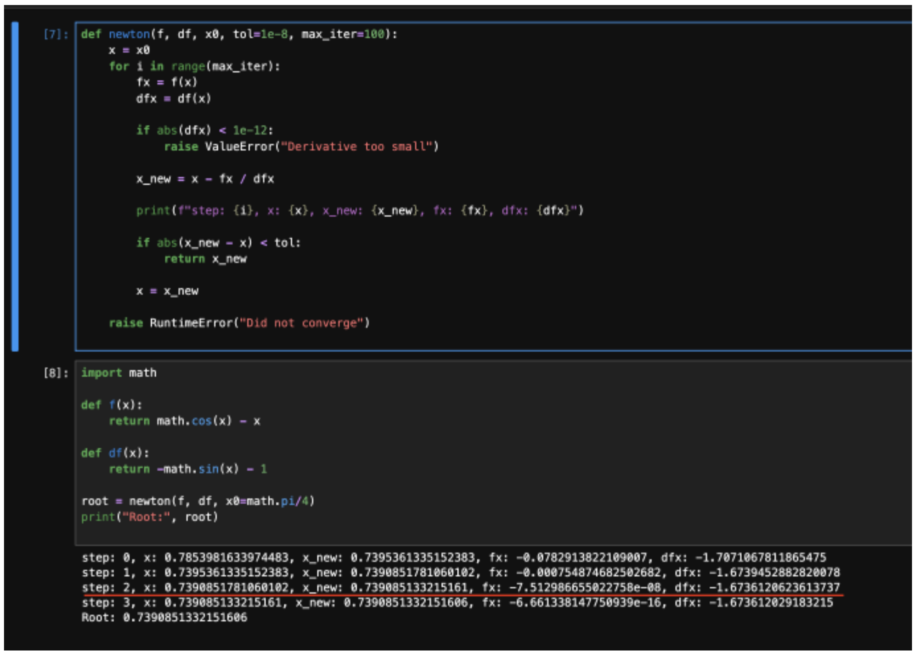
Fixed point

We want to solve $x = \cos(x)$. Define the iteration function: $g(x) = \cos(x)$. So that $x_{k+1} = g(x_k) = \cos(x_k)$.

Initial guess: $x_0 = \frac{\pi}{4} \approx 0.785398$.

$$\begin{aligned}
x_1 &= \cos(0.785398) \approx 0.707107\\
x_2 &= \cos(0.707107) \approx 0.764842\\
x_3 &= \cos(0.764842) \approx 0.722102\\
x_4 &= \cos(0.722102) \approx 0.750418\\
x_5 &= \cos(0.750418) \approx 0.731404\\
x_6 &= \cos(0.731404) \approx 0.744238\\
x^* &\approx 0.739085
\end{aligned}$$

Compare with Newton's Method



```
[7]: def newton(f, df, x0, tol=1e-8, max_iter=100):
         x = x0
         for i in range(max_iter):
             fx = f(x)
             dfx = df(x)

             if abs(dfx) < 1e-12:
                 raise ValueError("Derivative too small")

             x_new = x - fx / dfx

             print(f"step: {i}, x: {x}, x_new: {x_new}, fx: {fx}, dfx: {dfx}")

             if abs(x_new - x) < tol:
                 return x_new

             x = x_new

         raise RuntimeError("Did not converge")
```

```
[8]: import math

     def f(x):
         return math.cos(x) - x

     def df(x):
         return -math.sin(x) - 1

     root = newton(f, df, x0=math.pi/4)
     print("Root:", root)
```

```
step: 0, x: 0.7853981633974483, x_new: 0.7395361335152383, fx: -0.0782913822109007, dfx: -1.7071067811865475
step: 1, x: 0.7395361335152383, x_new: 0.7390851781060102, fx: -0.000754874682502682, dfx: -1.6739452882820078
step: 2, x: 0.7390851781060102, x_new: 0.739085133215161, fx: -7.512986655022758e-08, dfx: -1.6736120623613737
step: 3, x: 0.739085133215161, x_new: 0.7390851332151606, fx: -6.661338147750939e-16, dfx: -1.673612029183215
Root: 0.7390851332151606
```

Note

Fixed point iteration: converges slowly oscillating around the true root needs many iterations to stabilize. Newton's method converges much faster. By iteration 2, it already gives the root to 6 decimal places.

3.3 Convergence

Definition 3.6. Convergence

Let $e_k = [x_k - x^*]$. We say that the sequence $\{x_k\}$ converges to x^* with order q if:

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^q} = \mu = \text{constant}, \quad \mu > 0$$

(If $q = 1$, it is required that $\mu < 1$ for convergence.)

Example

- $q = 1$ (linear convergence)
- $q = 2$ (quadratic convergence)

Convergence of Bisection Method

$$\begin{aligned}|b_k - a_k| &= \frac{1}{2}|b_{k-1} - a_{k-1}| \\ &= \left(\frac{1}{2}\right)^k |b_0 - a_0|\end{aligned}$$

Recall: $|x_k - x^*| < |b_k - a_k| = \left(\frac{1}{2}\right)^k |b_0 - a_0|$.

Therefore, $x_k \to x^*$ as $k \to \infty$.

Note

$e_{k+1} \not< e_k$ in general for bisection, so convergence is a bit more erratic. For example x_k could happen to be closer to x^* than x_{k+1} on a particular iteration. Even though the $k+1$ interval width is smaller than k's.

But on average, $e_{k+1} \approx 0.5 e_k$.

3.3.1 Convergence of Fixed Point Iteration

Theorem 3.2 Contraction Mapping (Banach) Fixed Point Theorem

Let $g(x)$ be continuous on $[a,b]$. Assume $a \leq g(x) \leq b$ for all $x \in [a,b]$. (i.e. $g([a,b]) \subseteq [a,b]$). Then $x = g(x)$ has at least one solution.

Proof

Let $h(x) = g(x) - x$. Then $h(x)$ is continuous and $h(a) = g(a) - a \geq 0$, $h(b) = g(b) - b \leq 0$. By the Intermediate Value Theorem, there exists $c \in [a,b]$ such that $h(c) = 0 \implies g(c) - c = 0 \implies c$ is a fixed point of g.

$\square$

Definition 3.7. Let $g(x)$ be continuous and bounded on $[a,b]$. Then $g(x)$ is a **contraction** on $[a,b]$ if $0 \leq \lambda \leq 1$. s.t.

$$|g(x) - g(y)| \leq \lambda |x - y|, \quad \forall x, y \in [a,b]$$

Essentially, this is saying intervals within $[a, b]$ have their length decreased when you apply $g(x)$ to them.

Theorem 3.3

Suppose that:

1. $g(x)$ is continuous on $[a, b]$,
2. $g([a, b]) \subset [a, b]$,
3. $g(x)$ is a contraction on $[a, b]$.

Then the following is true;

1. $x = g(x)$ has a unique solution.
2. The sequence of iterates $x_{k+1} = g(x_k)$ converges to x^* for any $x_0 \in [a, b]$.

Proof

Existence follows from the previous theorem uniqueness: suppose α and β are two fixed points then

$$|\alpha - \beta| = |g(\alpha) - g(\beta)| \leq \lambda|\alpha - \beta|$$

so, $(1 - \lambda)|\alpha - \beta| \leq 0$ since $0 \leq \lambda < 1$, this implies that $|\alpha - \beta| = 0 \implies \alpha = \beta$.

Convergence:

$$e_k = |x^* - x_k| = |g(x^*) - g(x_{k-1})|$$

so, $e_k \leq \lambda|x^* - x_{k-1}| = \lambda e_{k-1}$. By induction,

$$e_k \leq \lambda^k e_0$$

Since $0 \leq \lambda < 1$, $\lambda^k \to 0$ as $k \to \infty$, so $e_k \to 0$.

□

Theorem 3.4

Let $g(x)$ be continuously differentiable on out interval $[a, b]$. Assume $g([a, b]) \subseteq [a, b]$ and

$$\lambda = \max_{a \leq x \leq b} |g'(x)| < 1.$$

Then

1. $g(x)$ has a unique fixed point $x^* \in [a, b]$.
2. $x_k \to x^*$ for $x_0 \in [a, b]$
3. $\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = g'(x^*)$

Proof

By Taylor expansion $g(x) = g(y) + (x - y)g'(y)$, where y is between x and y.

$$\begin{gathered} g(x) - g(y) = (x - y)g'(y) \\ |g(x) - g(y)| \le |x - y| \max_{a\le y\le b} |g'(y)| = \lambda|x - y| \end{gathered}$$

Then by Definition 3.7, $g(x)$ is a contraction on $[a, b]$.

Finally,

$$x^* - x_{k+1} = g(x^*) - g(x_k) = (x^* - x_k)g'(y_k)$$

Where y_k is between x^* and x_k.

Since $x_k \to x^*$, we also have $y_k \to x^*$. Thus,

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = \lim_{k\to\infty} g'(y_k) = g'(x^*)$$

Note: If $g'(x^*) \neq 0$, then fixed point iteration has **linear convergence**.

□

Example

$g(x) = x^2 - 2$	$g'(x) = 2x$	$g'(2) = 4 > 1$ (diverges)
$g(x) = \sqrt{x+2}$	$g'(x) = \frac{1}{2\sqrt{x+2}}$	$g'(2) = \frac{1}{4} < 1$ (converges)
$g(x) = 1 + \frac{2}{x}$	$g'(x) = -\frac{2}{x^2}$	$g'(2) = -\frac{1}{2} < 1$ (converges)

For $f(x) = x^2 - x - 2$, one root is $x^* = 2$. The possible iteration functions are:

3.3.2 Convergence of Newton's Method

Theorem 3.5

Suppose $f(x^*) = 0$, $x^* \in [a,b]$, $f'(x^*) \neq 0$, and f, f', f'' are bounded continous functions on $[a,b]$.

Then $\exists \delta$ s.t. if x_0 satisfies $x_0 - x^* < \delta$, then Newton's Iteration converges:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Recall

Mean Value Theorem: If g is continuous on $[x, x^*]$ and differentiable on (x, x^*), then $g(x) - g(x^*) = g'(y)(x - x^*)$ for some y in between x and x^*.

Proof

Consider newton as a fixed point iteration by writing $g(x) = x - \frac{f(x)}{f'(x)}$.

Then

$$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}$$

Since $f(x^*) = 0$, then $g'(x^*) = 0$.

Therefore, $\exists \delta$ such that $|g'(x)| \leq \lambda < 1$ for all x satisfying $|x - x^*| < \delta$.

Next, must show $g([x^* - \delta, x^* + \delta]) \subseteq [x^* - \delta, x^* + \delta]$.

Consider x with $|x - x^*| < \delta$. By the MVT (Recall 3.5), there exists y between x and x^* such that

$$|g(x) - g(x^*)| = |g'(y)|\,|x - x^*|.$$

Then $|g(x) - x^*| < |x - x^*| < \delta$, since $|g'(y)| \leq 1$.

Therefore, $g(x) \in [x^* - \delta, x^* + \delta]$.

Convenience follows from the theorem 3.4. So, newton converges if $f(x)$ is "nice" and x_0 is sufficiently close to x^*.

$\square$

Note

For the **convergence rate** we need to consider error ratios between k steps.

By Taylor expansion

$$0 = f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + \frac{f''(\xi_k)}{2}(x^* - x_k)^2$$

for ξ_k between x_k and x^*.

$\implies x^* - x_k + \frac{f(x_k)}{f'(x_k)} = -\frac{f''(\xi_k)}{2f'(x_k)}(x^* - x_k)^2$

But by definition of newton: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

Subtract both sides from x^*:

$$x^* - x_{k+1} = x^* - x_k + \frac{f(x_k)}{f'(x_k)} = -\frac{f''(\xi_k)}{2f'(x_k)}(x^* - x_k)^2$$

So, $e_{k+1} = C_k e_k^2$ where $C_k = -\frac{f''(\xi_k)}{2f'(x_k)}$.

(This hints at **quadratic convergence**)

Theorem 3.6

Assume f, f', f'' are continuous and $f'(x^*) = 0.$, and $f(x^*) = 0$, and $f'(x^*) \neq 0$. If x_0 is sufficiently close to x^*, then $x_k \to x^*$ and

$$\lim_{k\to\infty} \frac{e_{k+1}}{(e_k)^2} = -\frac{f''(x^*)}{2f'(x^*)}$$

i.e. Quadratic Convergence.

Proof

Let $I = [x^* - \delta, x^* + \delta]$, and define

$$M = \max_{x\in I} \frac{|f''(x)|}{2\min_{x\in I}|f'(x)|}$$

Then we have $|e_1| \le M|e_0|^2$. Also, $M|e_1| \le (M|e_0|)^2$.

Now pick x_0 such that $|e_0| \le \epsilon$, Such that $M|e_0| < 1$.

Then $M|e_1| \le (M|e_0|)^2 < 1^2$. So, $M|e_1| < 1$ too.

Squaring a nonnegative quality less than 1 gives a smaller or equal values.

So, $M|e_1| \le (M|e_0|)^2$ implies that $M|e_1| \le M|e_0|$. So, $|e_1| \le |e_0| \le \epsilon$.

Therefore, $|x_1 - x^*| \le \epsilon \implies x_1 \in I$.

By induction, $x_k \in I$ for all k. Similarly $|e_k| \le M|e_{k-1}|^2$.

Finally,

$$M|e_k| \le (M|e_{k-1}|)^2 \le (M|e_0|)^{2^k}$$

$\implies |e_k| \le \frac{1}{M}(M|e_0|)^{2^k}$.

Since $M|e_0| < 1$, this bound goes to 0 as $k \to \infty$. So, $|e_k| \to 0$.

Since y is between x_k and x^*,

$$\lim_{k\to\infty} \frac{e_{k+1}}{(e_k)^2} = -\lim_{k\to\infty} \frac{f''(\xi_k)}{2f'(x_k)} = -\frac{f''(x^*)}{2f'(x^*)} = \text{constant}$$

□

Note

- x_0 must be sufficiently close to x^*
- If newton is converging and x_k has s correct digits, then x_{k+1} has approximately $2s$ correct digits.
- Newton's method does not always converge quadratically consider the case where $f'(x^*) = 0$.

Method	**Does the method converge**
Bisection	Yes, guaranteed
Fixed Point Iteration	Depending on $g(x)$ and x_0
Newton's Method	Depending on $f(x)$ and x_0
Secant Method	Depending on $f(x)$ and x_0 and x_1

Method	**Speed of Convergence**	**Require knowledge of f'**
Bisection	Linear	No
Fixed Point Iteration	Linear / Superlinear	No
Newton's Method	Quadratic	Yes
Secant Method	$q \approx \frac{1}{2}(1+\sqrt{5})$	No

CHAPTER 4

Numerical Linear Algebra

Study of algorithms for performing linear algebra operations **Numerically**.

- Matrix/vector arithmetic
- Solving linear systems of equations
- Taking norms
- Factoring & inverting matrices
- Finding eigenvalues & eigenvectors

Definition 4.1. A **linear system** can be represented as

$$A\vec{x} = \vec{b},$$

where $A \in \mathbb{R}^{n\times n}$ is a matrix, $x \in \mathbb{R}^n$ is the vector of unknowns, and $b \in \mathbb{R}^n$ is the right-hand side vector.

Note

For $\vec{x}$ computationally, where we desire

- Accurate solution: we must make sure this a well-conditioned problem, and we have a stable algorithm
- Efficient algorithm: we want to minimize the number of operations

Theorem 4.1 Existence and Uniqueness

Consider $A\vec{x} = \vec{b}$.

- $\det(a) \neq 0$ (A has linear independent rows/columns; or A is invertible) if and only if $\vec{x} = A^{-1}\vec{b}$ exists and is unique.
- If $\det(A) = 0$, then
  - If $\vec{b} \in \text{range}(A)$, then there are infinitely many solutions.
  - If $\vec{b} \notin \text{range}(A)$, then there is no solution.

Note

The vector space V is said to be **Linearly dependent** if there exists scalars $a_1, a_2, \ldots, a_n$ not all zero such that

$$a_1\vec{v_1} + a_2\vec{v_2} + \ldots + a_n\vec{v_n} = \vec{0}.$$

If $a_1 \neq 0$, then we can write

$$\vec{v_1} = -\frac{a_2}{a_1}\vec{v_2} - \ldots - \frac{a_n}{a_1}\vec{v_n},$$

Thus, $\vec{v_1}$ is a linear combination for the remaining vectors.

A sequence of vectors $(\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n})$ is said to be linearly independent if the equation $a_1\vec{v_1} + a_2\vec{v_2} + \ldots + a_n\vec{v_n} = \vec{0}$ only has the trivial solution $a_1 = a_2 = \ldots = a_n = 0$

Column space: The column space of a matrix A is the span of its columns.

$$\text{Col}(A) = \text{span}\{\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}\}$$

Note

Why is it not a good idea to represent solution to $A\vec{x} = \vec{b}$ as $\vec{x} = A^{-1}\vec{b}$, where A^{-1} is the inverse of A (if it exists)?

Computing A^{-1} explicitly is computationally expensive and numerically unstable. Instead, it is better to use methods like Gaussian elimination or LU decomposition to solve the system directly without computing the inverse.

Definition 4.2. A matrix $A \in \mathbb{R}^{n\times n}$ with components a_{ij} is said to be **upper-triangular** if $a_{ij} = 0$ for all $i > j$; Similarly, A is said to be **lower-triangular** if $a_{ij} = 0$ for all $i < j$. A is triangular if is either upper-triangular or lower-triangular.

Warning

$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 5 \end{bmatrix}$ is not triangular. (It's not square)

4.1 Gaussian Elimination

Definition 4.3. **Gaussian elimination** is used to solve a linear system. Gaussian elimination may be performed in two phases:

1. Reduce the matrix A to upper triangular form.
2. Solve the reduced system by backward substitution.

4.2 Constructing LU Decomposition

We may write

$$(M^{(3)}M^{(2)}M^{(1)}) \cdot A = U$$

Let $U = A^{(4)}$, then

$$\begin{aligned} A &= (M^{(3)}M^{(2)}M^{(1)})^{-1} \cdot U \\ &= (M^{(1)})^{-1}(M^{(2)})^{-1}(M^{(3)})^{-1} \cdot U \\ &= L \cdot U \end{aligned}$$

(Note: $(AB)^{-1} = B^{-1}A^{-1}$)

Where $L = (M^{(1)})^{-1}(M^{(2)})^{-1}(M^{(3)})^{-1}$.

We define the matrix L_j as the inverse of the elimination matrix M_j.

Properties of matrices M_j and L_j:

- **Inversion Property:** L_j can be obtained from M_j by swapping the signs of the off-diagonal entries.

  Example, consider the matrix M_3.

$$M_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 25 & 1 \end{bmatrix} \implies L_3 = (M_3)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -25 & 1 \end{bmatrix}$$

- **Combination Property:** In general

$$L = \prod_{j=1}^{n-1} L_j = L_1 L_2 \cdots L_{n-1}$$

L can be obtained from the L_j by placing all the off-diagonal elements of the matrices L_j in the corresponding position in L.

Definition 4.4. L is called a **lower triangular matrix** with unit diagonal if and only if the matrix elements of L vanish above the diagonal and are 1 on the diagonal (i.e. $l_{ii} = 1$ for $i = 1, 2, \ldots, n$).

We may write the LU decomposition of A as

$$A = LU$$

with L unit lower triangular matrix and U upper triangular matrix.

For any $A \in \mathbb{R}^{n\times n}$, we obtain the **LU decomposition** of A by

$$A = (M^{(1)})^{-1}(M^{(2)})^{-1} \cdots (M^{(n-1)})^{-1} \cdot U = LU$$

Consider a linear system $A\vec{x} = \vec{b}$.

- Phase 1: Decompose $A = LU$ so we may write the linear system as $LU\vec{x} = \vec{b}$.
- Phase 2: Solve $L\vec{y} = \vec{b}$ for $\vec{y}$ by forward substitution.
- Phase 3: Solve $U\vec{x} = \vec{y}$ for $\vec{x}$ by back substitution.

Pivoting

Consider the following system:

$$\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

The entry in the (1,1) position is zero, which means the first pivot is zero. Since LU decomposition requires dividing by the pivor, the algorithm fails.

Strategy: Find the row with the largest magnitude entry in the current column beneath the current row, and swap those rows if larger than the current entry

$$\begin{bmatrix} 0 & 2 & 3 \\ 1 & 4 & 2 \\ -3 & 2 & -1 \end{bmatrix} \xrightarrow{R_3 \leftrightarrow R_1} \begin{bmatrix} -3 & 2 & -1 \\ 1 & 4 & 2 \\ 0 & 2 & 3 \end{bmatrix}$$

We immediately swap the first and third rows, since $|-3| > |1| > 0$.

Definition 4.5. $P \in \mathbb{R}^{n\times n}$ is a **permutation matrix** if and only if P is obtained from the unit matrix I_n by swapping any number of rows.

Theorem 4.2

For all $A \in \mathbb{R}^{n\times n}$, there exists a permutation matrix P, a unit lower triangular matrix L, and an upper triangular matrix U (all type $\mathbb{R}^{n\times n}$) such that $PA = LU$.

Corollary

If A is **nonsingular** then $A\vec{x} = \vec{b}$ can be solved by the LU decomposition algorithm applied to PA.

$A\vec{x} = \vec{b} \implies$ from previous theorem: $PA\vec{x} = P\vec{b}$ Again from previous theorem: $PA = LU$. We may write $LU\vec{x} = P\vec{b}$. Thus, we may simply apply forward and backward solvers to solve the system by LU decomposition. The backward and forward solve steps will not lead to division by zero, as L and U do not have any zero entries in the diagonal:

$$\begin{aligned} PA = LU &\implies \det(P)\det(A) = \det(L)\det(U) \\ &\implies \det(A) = \frac{\det(L)\det(U)}{\det(P)} \\ &\implies \prod_{i=1}^{n} u_{ij} \neq 0 \end{aligned}$$

Note: A is nonsingular $\neq 0$, $\det(L) = 1$, $\det(P) = \pm 1$, so $\det(U) \neq 0$.

4.2.1 Algorithm and Computational Cost

We aim to determine the (asymptotic) computational cost for solving an $n \times n$ system.

We measure cost in FLOPs (floating-point operations): the total count of additions, subtractions, multiplications, and divisions.

Note: FLOPs can refer either to the total number of operations or to floating-point operations per second.

Phase 1: Compute the LU decomposition, A = LU . The pseudocode for this algorithm is as follows:

```
% LU-Decomposition
l = diag(1)
u=A
for p = 1:n-1
    for r = p+1:n
        m = -u(r,p)/u(p,p)
        u(r,p) = 0
        for c = p+1:n
            u(r, c) = u(r, c) + m * u(p, c)
        end for
        l(r,p) = -m
    end for
end for
```

- Count A:
$$W = \left[\frac{1}{3}n^3 + O(n^2)\right] A$$
- Count M:
$$W = \left[\frac{1}{3}n^3 + O(n^2)\right] M$$

Total count: $W = \frac{2}{3}n^3 + O(n^2)$ flops.

Phase 2: We solve $L\vec{y} = \vec{b}$ by forward substitution

```
% Forward Substitution
y=b
for r = 2:n
    for c = 1:r-1
        y(r) = y(r) - L(r,c) * y(c)
    end for
end for
```

Total count: $W = n^2 + O(n)$ flops.

Phase 3: We solve $U\vec{x} = \vec{y}$ by backward substitution

```
% Backward Substitution
x=y
for r = n-1:-1:1
    for c = r+1:n
        x(r) = x(r) - U(r,c) * x(c)
    end for
    x(r) = x(r)/U(r,r)
end for
```

Total count: $W = n^2 + O(n)$ flops.

4.3 Determinant

Recall

Determinant of a matrix $A \in \mathbb{R}^{n\times n}$ is given by

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}) \quad \text{for fixed i}$$

where A_{1j} is the $(n-1)\times(n-1)$ matrix obtained by deleting the first row and j-th column of A.

Proposition

The following identities hold for determinants:

- $\det(BC) = \det(B)\det(C), (B, C \in \mathbb{R}^{n\times n})$
- $U \in \mathbb{R}^{n\times n}$ is upper triangular $\implies \det(U) = \prod_{i=1}^{n} u_{ii}$
- $L \in \mathbb{R}^{n\times n}$ is lower triangular $\implies \det(L) = \prod_{i=1}^{n} l_{ii}$
- $P \in \mathbb{R}^{n\times n}$ permutation matrix $\implies \det(P) = \pm 1$

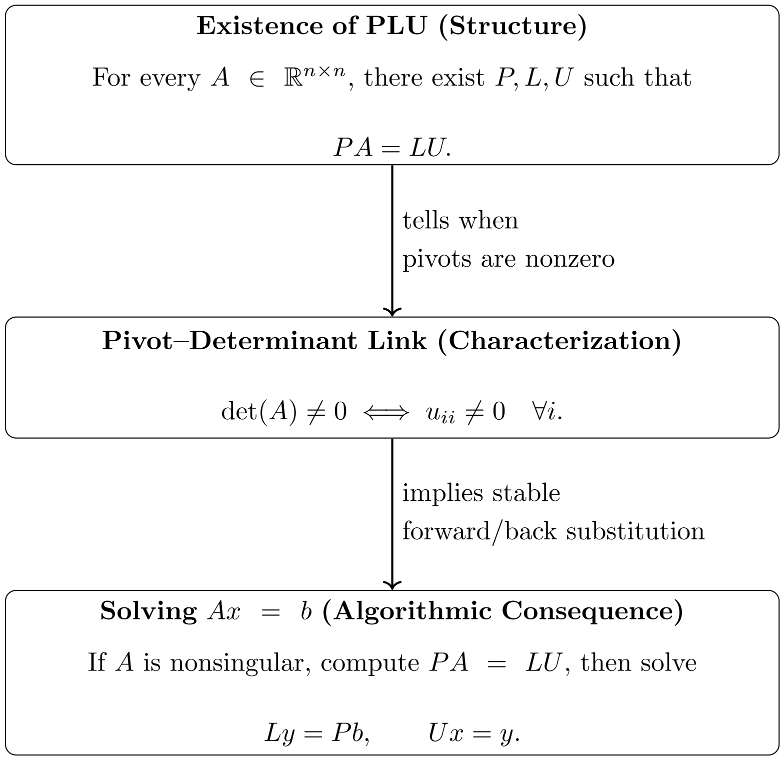
Recall

The algorithm for the LU decomposition can only be performed if there are no divisions by zero.

Proposition

Consider a matrix $A \in \mathbb{R}^{n\times n}$. Then $\det(A) \neq 0$ if and only if decomposition $PA = LU$ has $u_{ii} \neq 0, \forall i$.

4.4 Connection Between PLU Existence, Pivot Condition, and Solvability



Existence of PLU (Structure)

For every $A \in \mathbb{R}^{n\times n}$, there exist P, L, U such that

$$PA = LU.$$

tells when pivots are nonzero

Pivot–Determinant Link (Characterization)

$$\det(A) \neq 0 \iff u_{ii} \neq 0 \quad \forall i.$$

implies stable forward/back substitution

Solving $Ax = b$ (Algorithmic Consequence)

If A is nonsingular, compute $PA = LU$, then solve

$$Ly = Pb, \qquad Ux = y.$$

4.5 Condition and Stability

Norms are measurements of "size"/magnitude for vectors or matrices.

Recall

Definition of norm 2.5

Properties of norms:

- If the norm is zero, then the vector must be the zero-vector.

$$||x|| = 0 \implies x_i = 0 \quad \forall i$$

- The norm of a scaled vector must satisfy

$$||\alpha x|| = |\alpha| \cdot ||x|| \quad \text{for scalar } \alpha$$

- The triangle inequality must hold

$$||x + y|| \leq ||x|| + ||y||$$

4.5.1 Norms for Matrices

Definition 4.6. The **natural matrix p-norm** for $p = 1, 2, \infty$ is defined by

$$||A||_p = \max_{||\vec{x}|| \neq 0} \frac{||A\vec{x}||_p}{||\vec{x}||_p}$$

Proposition

$$||A\vec{x}||_p \leq ||A||_p \cdot ||\vec{x}||_p$$

Proposition

Consider a matrix $A \in \mathbb{R}^{n \times n}$ with elements a_{ij}. Then,

- $||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$ (maximum column sum)
- $||A||_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$ (maximum row sum)

Definition 4.7. Matrix 2-norm

Using the vector 2-norm, we get the matrix 2-norm (spectral norm):

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

Note: if λ_i are the eigenvalues of $A^T A$, then

$$||A||_2 = \max_i \sqrt{\lambda_i}$$

Proposition

Consider matrices $A, B \in \mathbb{R}^{n\times n}$ with $p = 1, 2, \infty$. Then,

$$||A+B||_p \leq ||A||_p + ||B||_p$$

Proof

For any matrix norm $||\cdot|| = ||\cdot||_p$, we have

$$||(A+B)x|| = ||Ax + Bx|| \leq ||Ax|| + ||Bx||$$

from the vector triangle inequality. Then, by the above proposition, we have:

$$\begin{aligned} &\implies ||(A+B)x|| \leq ||A||\cdot||x|| + ||B||\cdot||x|| \\ &\implies \frac{||(A+B)x||}{||x||} \leq ||A|| + ||B|| \end{aligned}$$

Thus, by definition of the matrix norm, we get

$$||A+B||_p \leq ||A||_p + ||B||_p$$

$\square$

Proposition

$\|A\|$ is a norm. In particular:

1. $\|A\| = 0 \iff A_{ij} = 0 \quad \forall\, i, j$.
2. $\|\alpha A\| = |\alpha|\,\|A\| \quad$ for scalar α.
3. $\|A+B\| \leq \|A\| + \|B\|$.
4. $\|Ax\| \leq \|A\|\,\|x\|$.
5. $\|AB\| \leq \|A\|\,\|B\|$.
6. $\|I\| = 1$.

4.5.2 Conditioning of Linear Systems

Conditioning describes how sensitive a function's output is to changes in its input.

A problem's conditioning indicates its inherent difficulty to solve, **independent** of the algorithm or numerical method used.

Condition of the problem $A\vec{x} = \vec{b}$

Find $\vec{x}$ from $A\vec{x} = \vec{b}$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular and $\vec{b} \in \mathbb{R}^n$.

Assume $\vec{x} = \vec{f}(A, \vec{b})$. Let's consider the change $\Delta\vec{x}$, if we have input $A + \Delta A$ and $\vec{b} + \Delta\vec{b}$:

Example

Consider the following linear system:

$$\begin{bmatrix} 1 & 2 \\ 0.499 & 1.001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1.5 \end{bmatrix}$$

Solution of this system is $\vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Now, if we perturb the input data slightly to

$$\begin{bmatrix} 1 & 2 \\ 0.5 & 1.001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1.5 \end{bmatrix}$$

the solution becomes $\vec{x} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$, which is a significant change from the original solution.

Note

Condition of the problem $A\vec{x} = \vec{b}$

In order to examine the condition of the initial problem $P(A\vec{x} = \vec{b})$ we need to consider a slight perturbation on the input data A and $\vec{b}$,

$$(A + \Delta A)(\vec{x} + \Delta\vec{x}) = \vec{b} + \Delta\vec{b}$$

Definition 4.8. The condition number of a matrix A is $\kappa(A) = ||A|| \cdot ||A^{-1}||$.

To find κ_R(relative condition number), we consider three cases:

- $\Delta A = 0$ and $\Delta\vec{b} \neq 0$
- $\Delta\vec{b} = 0$ and $\Delta A \neq 0$
- $\Delta b \neq 0$ and $\Delta A \neq 0$

Proof

For case 1 we have:

$A(\vec{x} + \Delta\vec{x}) = \vec{b} + \Delta\vec{b} \implies A\vec{x} + A\Delta\vec{x} = \vec{b} + \Delta\vec{b}$

Since $A\vec{x} = \vec{b} \implies A\Delta\vec{x} = \Delta\vec{b} \implies \Delta\vec{x} = A^{-1}\Delta\vec{b}$

Now we take norm of both sides:

$$||\Delta\vec{x}|| = ||A^{-1}\Delta\vec{b}|| \leq ||A^{-1}|| \cdot ||\Delta\vec{b}|| \quad (1)$$

Now if we take the norm of both sides of $A\vec{x} = \vec{b}$, we have

$$||\vec{b}|| = ||A\vec{x}|| \leq ||A|| \cdot ||\vec{x}|| \implies ||\vec{b}|| \cdot ||A^{-1}|| \leq ||\vec{x}|| \quad (2)$$

Combining (1) and (2), we get

$$\frac{||\Delta\vec{x}||}{||\vec{x}||} \leq ||A|| \cdot ||A^{-1}|| \cdot \frac{||\Delta\vec{b}||}{||\vec{b}||}$$

By rearrange the above inequality, we get

$$\kappa_R = \frac{||\Delta\vec{x}||/||\vec{x}||}{||\Delta\vec{b}||/||\vec{b}||} \leq ||A|| \cdot ||A^{-1}||$$

Thus if $\kappa(A)$ is small, then our problem P is well-conditioned.

Also, for the second and third cases, again $\kappa_R \leq ||A|| \cdot ||A^{-1}||$.

$\square$

Proposition

For a matrix $A \in \mathbb{R}^{m\times n}$,

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \sqrt{\frac{\lambda_{\max}(A^TA)}{\lambda_{\min}(A^TA)}}$$

Example

Compute the condition number of the matrix $\begin{bmatrix} 1 & 2 \\ 0.500 & 1.001 \end{bmatrix}$

The eigenvalue of A^TA is $\lambda_1 = 1.6 \times 10^{-7}$ and $\lambda_2 = 6.25$, so

$$\kappa_2(A) = \sqrt{\frac{6.25}{1.6 \times 10^{-7}}} = 6250$$

Since $\kappa_2(A)$ is large, the matrix A is ill-conditioned, which means that small changes in the input can lead to large changes in the output when solving linear systems involving A.

Stability of the LU Decomposition Algorithm

Consider the mathematical problem defined by $z(x) = \frac{a}{x}$ with a as a constant

- The absolute condition number is

$$\kappa_{\text{abs}} = \left|\frac{\Delta z}{\Delta x}\right| \approx \left|\frac{dz(x)}{dx}\right| = \left|\frac{-a}{x^2}\right| = \frac{|a|}{x^2}$$

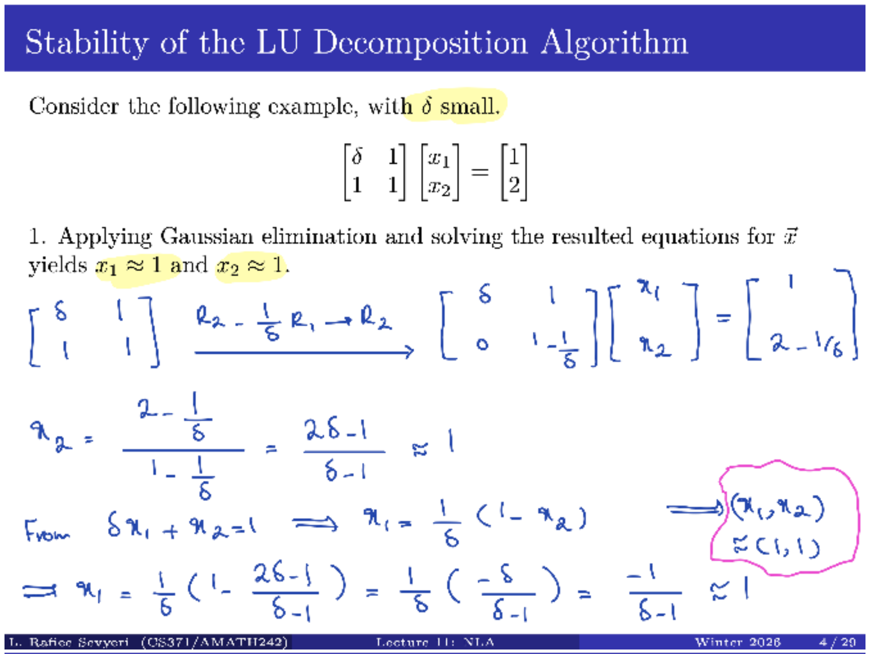
  Thus, if x is small then problem is ill-conditioned with respect to the absolute error.

- The relative condition number is computed as $\kappa_{\text{rel}} = 1$, so the problem is well-conditioned with respect to the relative error.

$$\kappa_{\text{rel}} = \frac{|\Delta z/z|}{|\Delta x/x|} \approx \left|\frac{dz(x)}{dx}\right| \cdot \left|\frac{x}{z(x)}\right| = \left|\frac{-a}{x^2}\right| \cdot \left|\frac{x^2}{a}\right| = 1$$

  The problem is ill-conditioned with respect to the absolute error but well-conditioned with respect to the relative error. This means that small changes in x can lead to large changes in $z(x)$ when x is small.

Example



Stability of the LU Decomposition Algorithm

Consider the following example, with δ small.

$$\begin{bmatrix}\delta & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}1\\ 2\end{bmatrix}$$

1. Applying Gaussian elimination and solving the resulted equations for $\vec{x}$ yields $x_1 \approx 1$ and $x_2 \approx 1$.

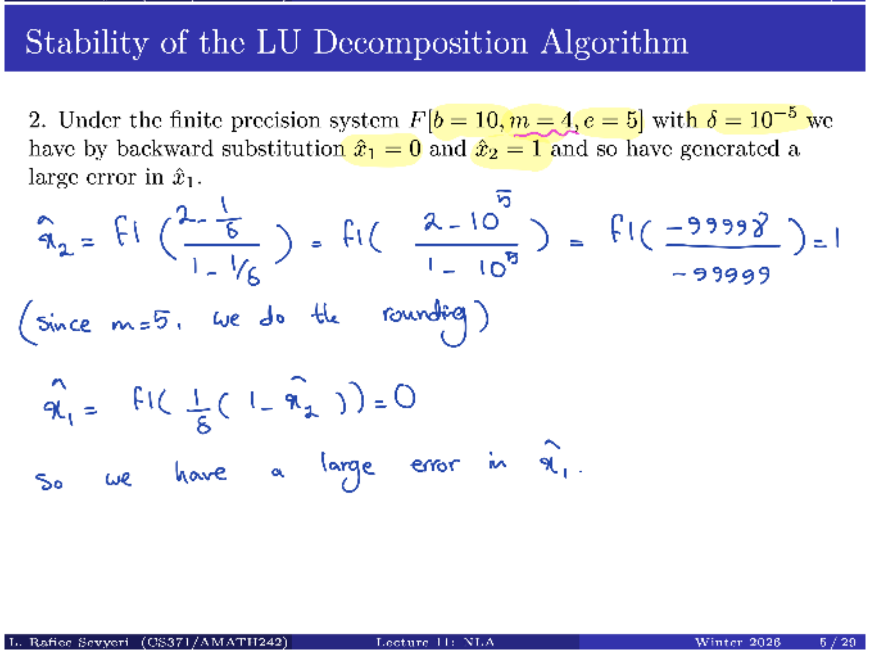
$$\begin{bmatrix}\delta & 1\\ 1 & 1\end{bmatrix} \xrightarrow{R_2-\frac{1}{\delta}R_1 \to R_2} \begin{bmatrix}\delta & 1\\ 0 & 1-\frac{1}{\delta}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}1\\ 2-1/\delta\end{bmatrix}$$

$$x_2 = \frac{2-\frac{1}{\delta}}{1-\frac{1}{\delta}} = \frac{2\delta-1}{\delta-1} \approx 1$$

From $\delta x_1 + x_2 = 1 \Longrightarrow x_1 = \frac{1}{\delta}(1-x_2)$ $\Longrightarrow \boxed{\begin{matrix}(x_1,x_2)\\ \approx(1,1)\end{matrix}}$

$$\Longrightarrow x_1 = \frac{1}{\delta}\left(1-\frac{2\delta-1}{\delta-1}\right) = \frac{1}{\delta}\left(\frac{-\delta}{\delta-1}\right) = \frac{-1}{\delta-1} \approx 1$$

L. Rafiee Sevyeri (CS371/AMATH242) Lecture 11: NLA Winter 2026 4 / 29



Stability of the LU Decomposition Algorithm

2. Under the finite precision system $F[b=10, m=4, e=5]$ with $\delta = 10^{-5}$ we have by backward substitution $\hat{x}_1 = 0$ and $\hat{x}_2 = 1$ and so have generated a large error in $\hat{x}_1$.

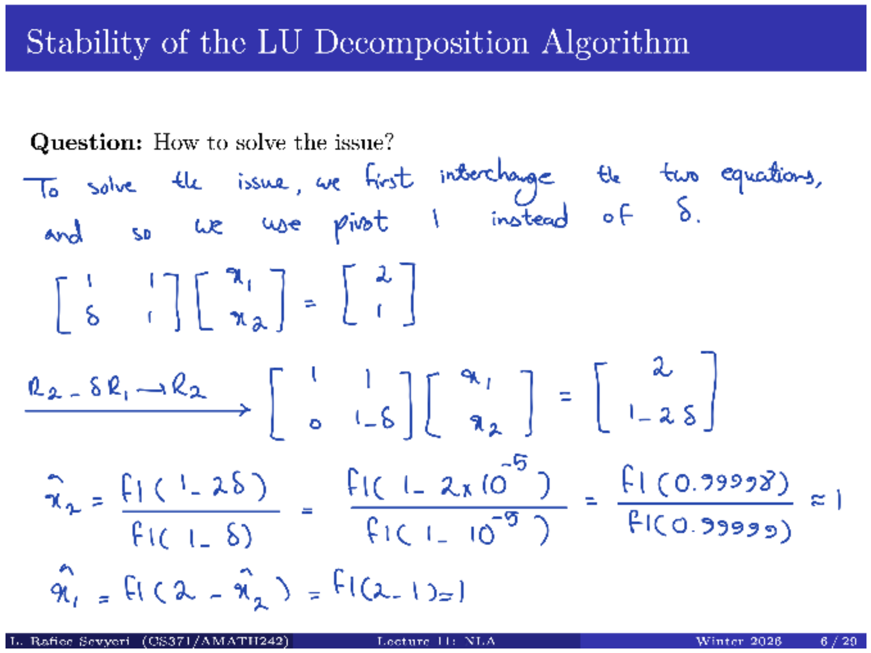
$$\hat{x}_2 = fl\left(\frac{2-\frac{1}{\delta}}{1-1/\delta}\right) = fl\left(\frac{2-10^5}{1-10^5}\right) = fl\left(\frac{-99998}{-99999}\right) = 1$$

(Since $m=5$, we do the rounding)

$$\hat{x}_1 = fl\left(\frac{1}{\delta}(1-\hat{x}_2)\right) = 0$$

So we have a large error in $\hat{x}_1$.

L. Rafiee Sevyeri (CS371/AMATH242) Lecture 11: NLA Winter 2026 5 / 29



Stability of the LU Decomposition Algorithm

Question: How to solve the issue?

To solve the issue, we first interchange the two equations, and so we use pivot 1 instead of δ.

$$\begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\xrightarrow{R_2 - \delta R_1 \to R_2} \begin{bmatrix} 1 & 1 \\ 0 & 1-\delta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1-2\delta \end{bmatrix}$$

$$\hat{x}_2 = \frac{fl(1-2\delta)}{fl(1-\delta)} = \frac{fl(1-2\times 10^{-5})}{fl(1-10^{-5})} = \frac{fl(0.99998)}{fl(0.99999)} \approx 1$$

$$\hat{x}_1 = fl(2-\hat{x}_2) = fl(2-1) = 1$$

L. Rafiee Sevyeri (CS371/AMATH242) Lecture 11: NLA Winter 2026 6 / 20

Note

LU decomposition with Partial Pivoting

In general, in order to minimize the error, we should rearrange the rows in every step of the algorithm so that we get the largest possible pivot element (in absolute value). This will give us the most stable algorithm for computing the solution to the linear system since we avoid divisions by small pivot elements. This approach is called LU decomposition with partial pivoting.

CHAPTER 5

Iterative Methods

Iterative Methods for solving $A\vec{x} = \vec{b}$

Definition 5.1. $A \in \mathbb{R}^{n\times n}$ is a **sparse matrix** if and only if the number of nonzero elements in A is much smaller than n^2.

Definition 5.2. $A \in \mathbb{R}^{n\times n}$ is **strictly diagonally dominant** if and only if

$$|a_{ii}| > \sum_{j\neq i, j=1}^{n} |a_{ij}|$$

for all rows of i.

Example

If $A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$, the non-zero element $2 << 3^2 = 9$ so A is sparse.

If $A = \begin{bmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{bmatrix}$, we have

- $|a_{11}| = 7 > 2 + 0 = \sum_{j\neq 1} |a_{1j}|$
- $|a_{22}| = 5 > 3 + 1 = \sum_{j\neq 2} |a_{2j}|$
- $|a_{33}| = 6 > 0 + 5 = \sum_{j\neq 3} |a_{3j}|$

so A is strictly diagonally dominant.

Proposition

A strictly diagonally dominant matrix is non-singular.

Note: In general if A is strictly diagonally dominant, the transpose of A does not necessarily retain the same property.

Example

If $A = \begin{bmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{bmatrix}$, $A^T = \begin{bmatrix} 7 & 3 & 0 \\ 2 & 5 & 5 \\ 0 & -1 & -6 \end{bmatrix}$

A is strictly diagonally dominant but A^T is not since $|a_{22}| = 5 \not> 2 + 5 = \sum_{j \neq 2} |a_{2j}|$.

Note

In following lectures, we will use $\vec{x}^{\text{new}}$ to denote the updated solution vector and $\vec{x}^{\text{old}}$ to denote the previous solution vector.

5.1 Jacobi and Gauss-Seidel Methods

Jacobi Method

We assume $A \in \mathbb{R}^{3\times 3}$ and construct a system of equations as follows:

$$\begin{aligned} a_{11}x_1^{\text{new}} + a_{12}x_2^{\text{old}} + a_{13}x_3^{\text{old}} &= b_1 \\ a_{21}x_1^{\text{old}} + a_{22}x_2^{\text{new}} + a_{23}x_3^{\text{old}} &= b_2 \\ a_{31}x_1^{\text{old}} + a_{32}x_2^{\text{old}} + a_{33}x_3^{\text{new}} &= b_3 \end{aligned}$$

Therefore,

$$x_i^{\text{new}} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{\text{old}} \right) \quad \text{for } i = 1, 2, 3$$

Gauss-Seidel

In this method we use the elements of $\vec{x}^{\text{new}}$ derived earlier in the same step and construct a linear system as:

$$\begin{aligned} a_{11}x_1^{\text{new}} + a_{12}x_2^{\text{old}} + a_{13}x_3^{\text{old}} &= b_1 \\ a_{21}x_1^{\text{new}} + a_{22}x_2^{\text{new}} + a_{23}x_3^{\text{old}} &= b_2 \\ a_{31}x_1^{\text{new}} + a_{32}x_2^{\text{new}} + a_{33}x_3^{\text{new}} &= b_3 \end{aligned}$$

Rearranging the equations, we get

$$x_i^{\text{new}} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{\text{new}} - \sum_{j>i} a_{ij}x_j^{\text{old}}\right) \quad \text{for } i = 1, 2, 3$$

Note

Note: The Gauss-Seidel method is more efficient than the Jacobi method since it uses the most up-to-date values of $\vec{x}$ in each iteration.

In matrix form presentation

Using the decomposition $A = A_L + A_D + A_U$, we can express the Jacobi and Gauss-Seidel methods in matrix form as follows:

- Jacobi method: $\vec{x}^{\text{new}} = A_D^{-1}(\vec{b} - (A_L + A_U)\vec{x}^{\text{old}})$
- Gauss-Seidel method: $\vec{x}^{\text{new}} = A_D^{-1}(\vec{b} - (A_L\vec{x}^{\text{new}} + A_D\vec{x}^{\text{new}}))$

$$A_L = \begin{bmatrix} 0 & 0 & 0 \\ * & 0 & 0 \\ * & * & 0 \end{bmatrix}, \quad A_D = \begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix}, \quad A_U = \begin{bmatrix} 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{bmatrix}$$

$$A = A_L + A_D + A_U = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

$$\begin{aligned} \vec{b} &= A\vec{x} = (A_L + A_D + A_U)\vec{x} \\ &= (A_L + A_U)\vec{x} + A_D\vec{x} \\ \Rightarrow A_D\vec{x} &= \vec{b} - (A_L + A_U)\vec{x} \end{aligned}$$

Now if A_D^{-1} exists, we can derive the Jacobi method as

$$\vec{x} = A_D^{-1}(\vec{b} - (A_L + A_U)\vec{x})$$

5.2 Convergence of Iterative Methods

Theorem 5.1

Consider $A\vec{x} = \vec{b}$ and let $\vec{x}^0$ be any starting vector. Let $\{\vec{x}^{(i)}\}_{i=1}^{\infty}$ be the sequence generated by either Jacobi or Gauss-Seidel iterative methods. Then if A is strictly diagonally dominant the sequence converge to the unique solution of the system $A\vec{x} = \vec{b}$.

Definition 5.3. The residual of a linear system $A\vec{x} = \vec{b}$ for some vector $\vec{u}$ is $\vec{r} = \vec{b} - A\vec{u}$.

As $\vec{r}^{(i)}$ (residual at step i) becomes smaller, $\vec{x}^{(i)}$ becomes closer to the solution of $A\vec{x} = \vec{b}$.

$$\frac{||\vec{r}^{(i)}||_2}{||\vec{r}^{(0)}||_2} \leq t$$

For an iterative approximation $\vec{u}$, the error is given by $\vec{e} = \vec{x} - \vec{u}$. This leads to the following relationship between the error and the residual:

$$A\vec{e} = A(\vec{x} - \vec{u}) = A\vec{x} - A\vec{u} = \vec{b} - A\vec{u} = \vec{r}$$

If we knew the error $\vec{e}$ for any approximation $\vec{u}$, we could write out the solution to the linear system directly:

$$\vec{x} = \vec{u} + (x - \vec{u}) = \underbrace{\vec{u}}_{\text{approximation}} + \underbrace{\vec{e}}_{\text{error}}$$

However, if error is unknown, we can instead use the residual:

$$\vec{x} = \vec{u} + A^{-1}\vec{r}$$

Note: Inverting A is an expensive operation.

Instead of using A^{-1}, we can choose a matrix B that is easy to invert such that B^{-1} is an approximation for A^{-1}, Then:

$$\vec{x} = \vec{u} + B^{-1}\vec{r}$$

Repeat this process iteratively, we get

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + B^{-1}(\vec{b} - A\vec{x}^{(i)})$$

Theorem 5.2

Consider the iterative method

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + B^{-1}(\vec{b} - A\vec{x}^{(i)})$$

with $\det(A) \neq 0$. If there exists a p-norm for which $||I - B^{-1}A||_p < 1$ then the iterative method will converge for any starting value $\vec{x}^{(0)}$. Convergence will then hold in any p-norm.

Proof

Consider the error $\vec{e}^{(i)} = \vec{x} - \vec{x}^{(i)}$. Then,

$$\begin{aligned} \vec{x}^{(i+1)} - \vec{x} &= \vec{x}^{(i)} - \vec{x} + B^{-1}(\vec{b} - A\vec{x}^{(i)}) \\ \vec{x}^{(i+1)} - \vec{x} &= -\vec{e}^{(i)} + B^{-1}(A\vec{x} - A\vec{x}^{(i)}) \\ \vec{e}^{(i+1)} &= \vec{e}^{(i)} - B^{-1}A\vec{e}^{(i)} \end{aligned}$$

Take norm:

$$||\vec{e}^{(i+1)}||_p = ||(I - B^{-1}A)\vec{e}^{(i)}||_p \leq ||I - B^{-1}A||_p \cdot ||\vec{e}^{(i)}||_p$$

If we apply this result recursively, we get

$$||\vec{e}^{(i+1)}||_p \leq ||I - B^{-1}A||_p^{i+1} \cdot ||\vec{e}^{(0)}||_p$$

Taking the limit as $i \to \infty$, we have

$$\lim_{i\to\infty} ||\vec{e}^{(i+1)}||_p \leq \lim_{i\to\infty} ||I - B^{-1}A||_p^{i+1} \cdot ||\vec{e}^{(0)}||_p = 0$$

where we have $||I - B^{-1}A||_p < 1$ by assumption. Therefore, $\vec{e}^{(i)} \to 0$ as $i \to \infty$, which means $\vec{x}^{(i)} \to \vec{x}$ as $i \to \infty$.

□

Determine B for the Jacobi and Gauss-Seidel methods:

- Jacobi method: $B^{-1} = A_D^{-1}$
- Gauss-Seidel method: $B^{-1} = (A_L + A_D)^{-1}$

Proof

If we rewrite the matrix form of Jacobi into standard form, we have:

$$\vec{x}^{(i+1)} = A_D^{-1}\left(\vec{b} - (A_L + A_U)\vec{x}^{(i)}\right)$$

We know $A = A_D + A_L + A_U$, so we can rewrite the above as:

$$\begin{aligned}
\vec{x}^{(i+1)} &= A_D^{-1}\left(\vec{b} - (A - A_D)\vec{x}^{(i)}\right)\\
&= A_D^{-1}\left(\vec{b} - A\vec{x}^{(i)} + A_D\vec{x}^{(i)}\right)\\
&= A_D^{-1}\vec{b} + (I - A_D^{-1}A)\vec{x}^{(i)}\\
\vec{x}^{(i+1)} &= \vec{x}^{(i)} + A_D^{-1}\left(\vec{b} - A\vec{x}^{(i)}\right)\\
B^{-1} &= A_D^{-1}
\end{aligned}$$

□

CHAPTER 6

Interpolation

6.1 Polynomial Interpolation

Definition 6.1.

Problem: $(n+1)$ discrete points $\{(x_i, f_i)\}_{i=0}^{n}$ are given with $x_i \neq x_j$ for $i \neq j$.
Objective: Defining a **continuous function** that interpolates the data: $y(x_i) = f_i$ for $0 \leq i \leq n$.

The algebraic polynomials:

$$y_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

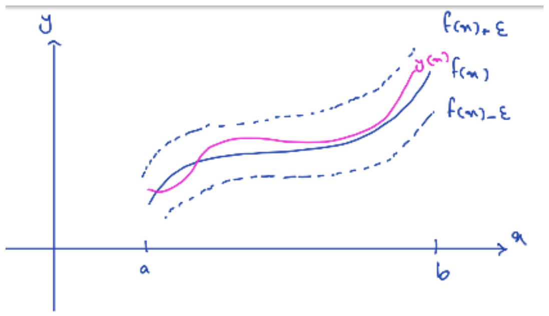
where n is non-negative integer and $a_0, a_1, \ldots, a_n$ are real numbers.

We will focus on polynomial interpolation because

- They are easy to be integrated and differentiated.
- They uniformly approximate continuous functions.

Theorem 6.1 Weierstrass Approximation Theorem

If $f(x)$ is a continuous function on $[a, b]$. For each $\epsilon > 0$, there exists a polynomial $y(x)$ such that $|f(x) - y(x)| < \epsilon$ for all $x \in [a, b]$.



Definition 6.2. Monomial Basis

To interpolate $n+1$ data points using polynomials, we denote Y_n as a set of all polynomials of degree at most n. In addition and scalar multiplication defined, Y_n forms a vector space with basis

$$\phi_j = x^j, j = 0, \dots, n$$

for which a given polynomial $y_n \in Y_n$ has the form

$$y_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Each a_j is a real number and is called the j-th **coefficient** of y_n, and x^j is called the j-th **monomial** of y_n.

Definition 6.3. Vandermonde Matrix

In the monomial base, the vector $\vec{a}$ of coefficients of the polynomial interpolating the data points (x_i, f_i), for $i = 0, 1, 2, \dots, n$ is given by the $(n+1) \times (n+1)$ linear system $V\vec{a} = \vec{f}$, where:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

A matrix in this form, whose columns are successive powers of some independent variable x is called Vandermonde Matrix.

Proposition

The determinant of V is given by

$$\det(V) = \prod_{0\le i<j\le n} (x_j - x_i)$$

Which is equivalent to

$$\begin{aligned}\det(V) =&(x_1 - x_0)\\ &(x_2 - x_0)(x_2 - x_1)\\ &\dots\\ &(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})\end{aligned}$$

Proof

Let V be the $(n+1)\times(n+1)$ Vandermonde matrix:

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}$$

Step 1: View $\det(V)$ as a polynomial in x_n. Expend $\det(V)$ along the last row (row n)

$$\begin{aligned}\det(V) &= \sum_{j=0}^{n} (-1)^{n+j} V_{nj} \det(V_{nj})\\ &= \sum_{j=0}^{n} (-1)^{n+j} x_n^j \det(V_{nj})\end{aligned}$$

where V_{nj} is the matrix obtained by deleting the n-th row and j-th column of V.

Each $\det(V_{nj})$ depends only on $x_0, x_1, \dots, x_{n-1}$, so it is independent of x_n. Therefore, $\det(V)$ is a polynomial in x_n of degree at most n.

Moreover, the coefficient of x_n^n comes only from the term $j = n$: $(-1)^{n+n} x_n^n \det(V_{nn}) = x_n^n \det(V_{nn})$, so the leading coefficient is $\det(V_{nn})$. Hence $\deg(\det(V)) = n$ and $\det(V) = \det(V_{nn})x_n^n +$ lower degree terms.

Step 2: Find the roots: If $x_n = x_k$ for some $k \in \{0, 1, \dots, n-1\}$, then row n equals row k, so $\det(V) = 0$. Thus, $\det(V)$, as a polynomial in x_n, has roots $x_0, x_1, \dots, x_{n-1}$. Since it has degree n, it must be of the form $\det(V) = b(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})$

for some constant b independent of x_n. From step 1, the leading coefficient of $\det(V)$ is $\det(V_{nn})$, therefore $b = \det(V_{nn}) \implies \det(V) = \det(V_{nn}) \prod_{k=0}^{n-1}(x_n - x_k)$. The matrix V_{nn} is exactly the $n \times n$ Vandermonde matrix built from $x_0, x_1, \ldots, x_{n-1}$. By induction,

$$\det(V_{nn}) = \prod_{0 \le i < j \le n-1} (x_j - x_i)$$

Multiplying the above two equations, we have

$$\det(V) = \prod_{0 \le i < j \le n-1} (x_j - x_i) \prod_{k=0}^{n-1} (x_n - x_k) = \prod_{0 \le i < j \le n} (x_j - x_i)$$

□

Theorem 6.2

The interpolating polynomial $y_n(x)$ exists and is unique

Proof

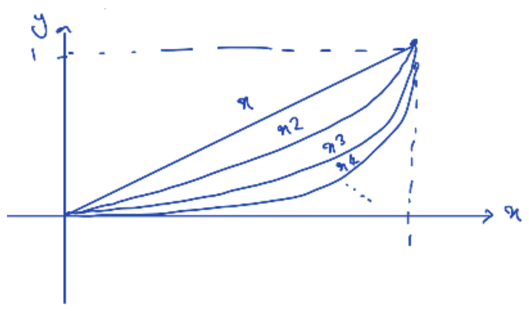
In order to have a unique solution for the linear system $V\vec{a} = \vec{f}$, V must be invertible $\implies \det(V) \neq 0$. We know $\det(V) = \prod_{0 \le j < i \le n}(x_j - x_i)$. Thus $\det(V) \neq 0 \iff x_i \neq x_j$ for $i \neq j$. Thus, by a standard result from linear algebra, we know the linear system has a unique solution (or y_n exists and is unique).

□

Note

Challenges of using monomial basis for polynomial interpolation

- Computational complexity to solve the linear system using standard solver (G.E) is of order $O(n^3)$, solvers with complexity of order $O(n^2)$ (for Vandermonde systems) use other polynomial representation.
- V is a very ill-conditioned matrix, since $\kappa_2(V)$ grows faster than exponentially as a function of n.



Note

Does this ill-conditioning prevent the computed polynomial from fitting the data point?

No, the computed polynomial will fit the data points because Gaussian elimination with partial pivoting will produce a small residual for the solution of the linear system in any case.

Question: Does this ill-conditioning prevent the computed polynomial from fitting the data point?

No, the computed

Note

By using different basis, both the conditioning of the linear system and the amount of computational work required to solve the linear system can be improved.

A change of basis still gives the **same unique interpolation polynomial**

Proof

If we have two interpolation polynomials

$$p_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

and

$$q_n(x) = b_0 + b_1x + b_2x^2 + \cdots + b_nx^n$$

Now their difference would be a **polynomial of degree at most** n

$$r_n(x) = (a_0 - b_0) + (a_1 - b_1)x + (a_2 - b_2)x^2 + \cdots + (a_n - b_n)x^n$$

If we evaluate $p_n - q_n$ at every data point $x_0, \ldots, x_n$, we have

$$\begin{aligned}(p_n - q_n)(x_i) &= p_n(x_i) - q_n(x_i) \\ &= f_i - f_i \\ &= 0\end{aligned}$$

This means that $p_n - q_n$ has $(n+1)$ distinct roots. But $p_n - q_n$ is of degree at most n, so it can only have at most n roots values it is the zero polynomial. Therefore, $p_n - q_n \equiv 0 \implies p_n \equiv q_n$.

So, what has changed is only the representation of that polynomial in a different basis, but the polynomial itself is the same.

□

6.1.1 Monomial Basis

Conditioning of monomial basis can be improved by shifting and scaling the independent variable

$$\phi_j(x) = \left(\frac{x-c}{d}\right)^j, \quad j = 0, 1, \ldots, n$$

where

$$c = \frac{x_1 + x_n}{2}, \quad d = \frac{x_n - x_1}{2}.$$

Then

$$\phi(x) = \frac{x - \frac{x_1 + x_n}{2}}{\frac{x_n - x_1}{2}} = \frac{2x - (x_1 + x_n)}{x_n - x_1}.$$

If $x = x_1$, then

$$\begin{aligned}\phi(x_1) &= \frac{2x_1 - (x_1 + x_n)}{x_n - x_1} \\ &= \frac{x_1 - x_n}{x_n - x_1} \\ &= -1\end{aligned}$$

If $x = x_n$, then

$$\begin{aligned}\phi(x_n) &= \frac{2x_n - (x_1 + x_n)}{x_n - x_1} \\ &= \frac{x_n - x_1}{x_n - x_1} \\ &= 1.\end{aligned}$$

Since ϕ is linear and increasing on $[x_1, x_n]$, it maps the interval $[x_1, x_n]$ exactly **onto** $[-1, 1]$.

This transformation also helps avoid overflow and underflow when computing the entries of the basis matrix or evaluating the interpolation polynomial.

However, the monomial basis, even after shifting and scaling the independent variable, still suffers from poor conditioning for large n.

6.2 Lagrange Interpolation

Definition 6.4. Determining a polynomial of degree one that passes through distinct points (x_0, f_0) and (x_1, f_1).

This polynomial is in the form $y_1(x) = a_0 + a_1 x$. Now considering $y(x_0) = f_0$ and $y(x_1) = f_1$,

From two points form of the line, we get:

$$y_1 - f_1 = \frac{f_0 - f_1}{x_0 - x_1}(x - x_1) = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x_1 - x_0} f_1$$

Rewriting the above equation, we have

$$y_1 = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x_1 - x_0} f_1$$

Definition 6.5. Lagrange Basis

The Lagrange basis functions are defined as

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

So the interpolation polynomial can be written as

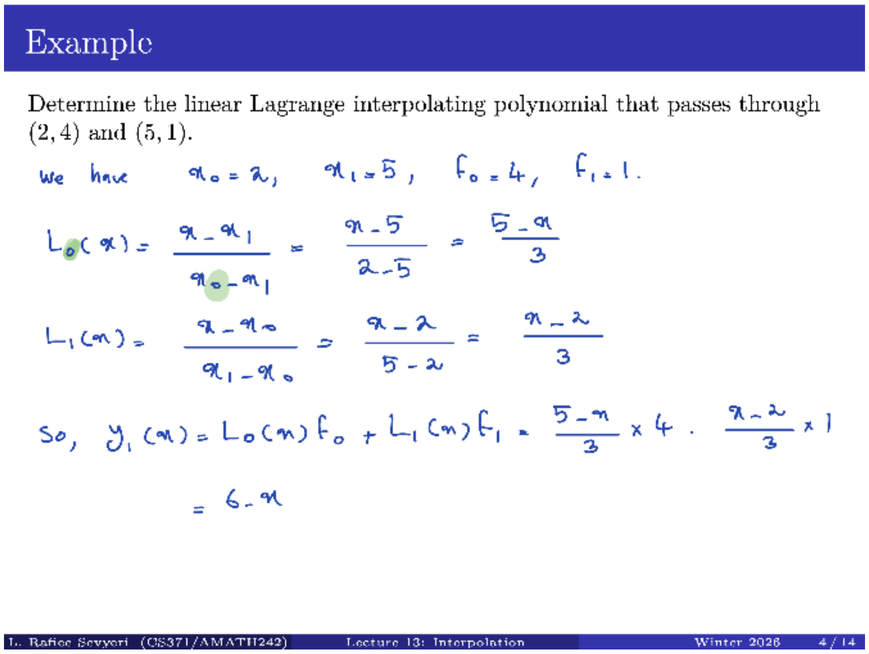
$$y_1(x) = L_0(x)f_0 + L_1(x)f_1$$

Note

Note that $L_0(x_0) = 1$, $L_1(x_1) = 1$, $L_0(x_1) = 0$, and $L_1(x_0) = 1$:

$$L_0(x_0) = \frac{x_0 - x_1}{x_0 - x_1} = 1, \quad L_1(x_1) = \frac{x_1 - x_0}{x_1 - x_0} = 1$$
$$L_0(x_1) = \frac{x_1 - x_1}{x_0 - x_1} = 0, \quad L_1(x_0) = \frac{x_0 - x_0}{x_1 - x_0} = 0$$

Example



Example

Determine the linear Lagrange interpolating polynomial that passes through $(2, 4)$ and $(5, 1)$.

We have $x_0 = 2$, $x_1 = 5$, $f_0 = 4$, $f_1 = 1$.

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} = \frac{x-5}{2-5} = \frac{5-x}{3}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0} = \frac{x-2}{5-2} = \frac{x-2}{3}$$

So, $y_1(x) = L_0(x)f_0 + L_1(x)f_1 = \frac{5-x}{3} \times 4 + \frac{x-2}{3} \times 1$

$= 6 - x$

L. Rafiee Sevyeri (CS371/AMATH242) Lecture 13: Interpolation Winter 2026 4 / 14

Theorem 6.3 Lagrange Interpolation Theorem

Given $n + 1$ distinct points $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$, there exists a unique polynomial of degree at most n that interpolates these points. This polynomial can be expressed in terms of the Lagrange basis functions as

$$y_n(x) = \sum_{j=0}^{n} L_j(x) f_j$$

where

$$L_j(x) = \prod_{\substack{0 \le m \le n \\ m \ne j}} \frac{x - x_m}{x_j - x_m}, \quad j = 0, 1, \ldots, n$$

Definition 6.6. Monomial basis or standard basis is the basis to form $\mathbb{Y}_n(x)$ as a vector space that contains all polynomials of degree at most n:

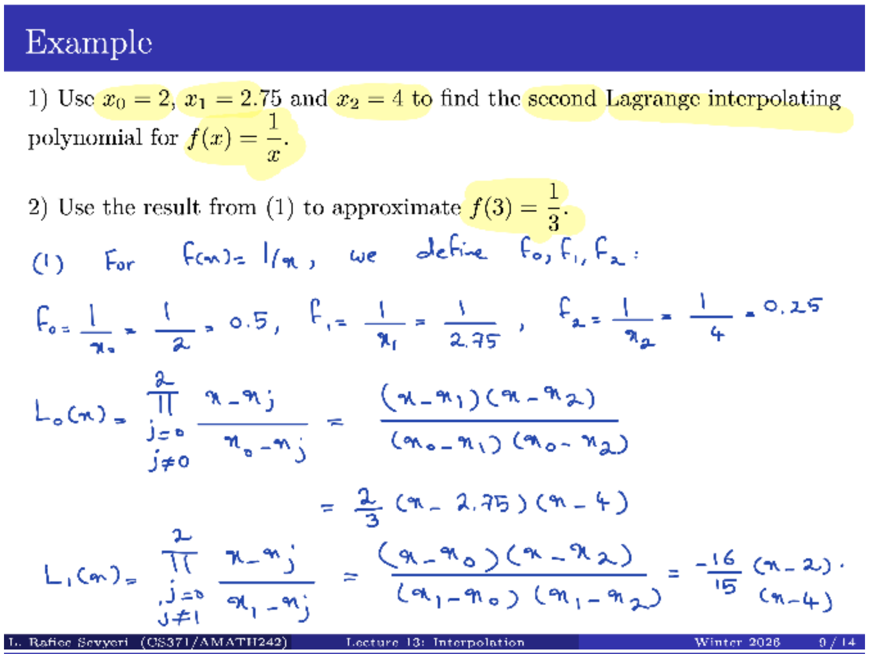
$$\mathbb{Y}_n(x) = \text{span}\{1, x, x^2, \ldots, x^n\}$$

The **Lagrange polynomial** form a different basis for $\mathbb{Y}_n(x)$:

$$B' = \{L_0(x), L_1(x), \ldots, L_n(x)\}$$

Since this is also a basis, we can write any polynomial $y_n(x)$ as a linear combination of the Lagrange polynomial $y_n(x) = \sum_{j=0}^{n} L_j(x) f_j$.

Example



Example

1) Use $x_0 = 2$, $x_1 = 2.75$ and $x_2 = 4$ to find the second Lagrange interpolating polynomial for $f(x) = \dfrac{1}{x}$.

2) Use the result from (1) to approximate $f(3) = \dfrac{1}{3}$.

(1) For $f(x) = 1/x$, we define f_0, f_1, f_2:

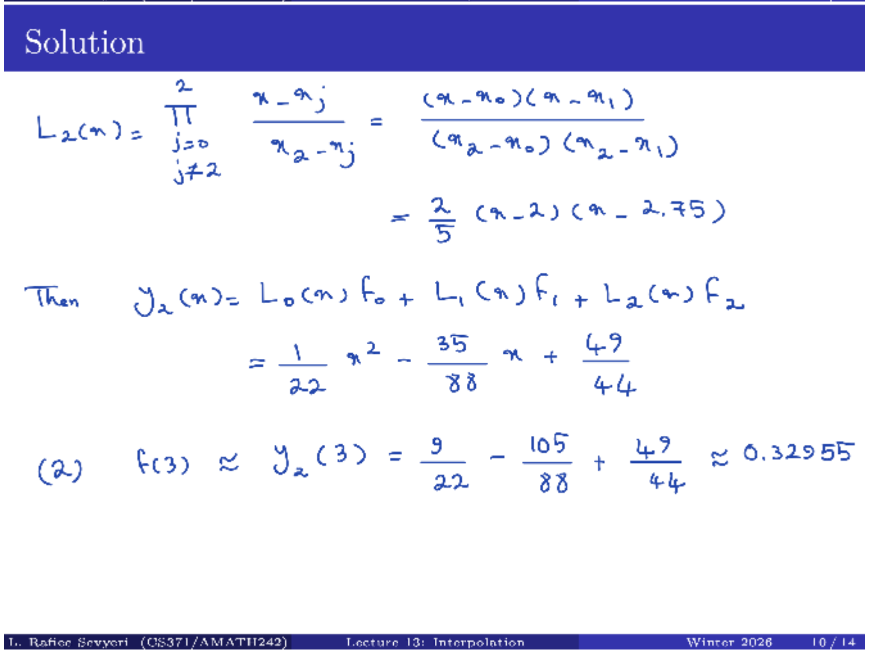
$$f_0 = \frac{1}{x_0} = \frac{1}{2} = 0.5, \quad f_1 = \frac{1}{x_1} = \frac{1}{2.75}, \quad f_2 = \frac{1}{x_2} = \frac{1}{4} = 0.25$$

$$L_0(x) = \prod_{\substack{j=0\\ j\neq 0}}^{2} \frac{x-x_j}{x_0-x_j} = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}$$

$$= \frac{2}{3}(x-2.75)(x-4)$$

$$L_1(x) = \prod_{\substack{j=0\\ j\neq 1}}^{2} \frac{x-x_j}{x_1-x_j} = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{-16}{15}(x-2)\cdot(x-4)$$

L. Rafiee Sevyeri (CS371/AMATH242) Lecture 13: Interpolation Winter 2026 9 / 14



Solution

$$L_2(x) = \prod_{\substack{j=0\\ j\neq 2}}^{2} \frac{x-x_j}{x_2-x_j} = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

$$= \frac{2}{5}(x-2)(x-2.75)$$

Then $y_2(x) = L_0(x)f_0 + L_1(x)f_1 + L_2(x)f_2$

$$= \frac{1}{22}x^2 - \frac{35}{88}x + \frac{49}{44}$$

(2) $f(3) \approx y_2(3) = \dfrac{9}{22} - \dfrac{105}{88} + \dfrac{49}{44} \approx 0.32955$

L. Rafiee Sevyeri (CS371/AMATH242) Lecture 13: Interpolation Winter 2026 10 / 14

Theorem 6.4

Suppose $x_0, x_1, \ldots, x_n$ are distinct numbers in the interval $[a, b]$ and f is a continuous and differentiable function. Then for each x in $[a, b]$, there exists a number $\xi(x)$ (generally unknown) between $x_0, x_1, \ldots, x_n$ and hence in (a, b) with

$$f(x) - y_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n)$$

where $y_n(x)$ is the Lagrange interpolation polynomial.

Note

Note that the error term for the Lagrange polynomial is quite similar to that for the Taylor polynomial. The nth Taylor polynomial about x_0 has an error term of the form:

$$\frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}$$

The Lagrange polynomial of degree n uses information at distinct points $x_0, x_1, \ldots, x_n$, and instead of $(x-x_0)^{n+1}$, its error formula uses the product of $n+1$ terms $(x-x_0)(x-x_1)\cdots(x-x_n)$.