MATH3312: Numerical Analysis

Jett Cheung

January 14, 2026

Lecture notes from 2025 Fall MATH3312: Numerical Analysis, given by Prof. Peng Zhichao at the Hong Kong University of Science and Technology in the academic year Fall 2025. This course covers computer arithmetic, root-finding, interpolation, numerical calculus and solving linear systems.

Disclaimer: This document will inevitably contain some mistakes – both simple typos and legitimate errors. Keep in mind that these are the notes of an undergraduate student in the process of learning the material himself, so take what you read with a grain of salt. If you find mistakes and feel like telling me, I will be grateful and happy to hear from you, even for the most trivial of errors. You can reach me by email, in English at mycheungai@conenct.ust.hk.

Contents

Chapter 1

Computer Arithmetic

01/09/2025

Lecture 1: Introduction, Floating Number and Round-Off Error

1.1 Floating Number

1.1.1 Decimal and Binary Numbers

In daily life, numbers are usually represented in decimals, with allowed digits $\{0, 1, \cdots, 9\}$. However, computer processes information in binary bits, with allowed bits $\{0, 1\}$.

Example 1.1. Binary $\to$ Decimal

$$11.101_2 \;=\; 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 3.625.$$

1.1.2 Floating Number Representation

In a binary floating-point system, numbers are represented as

$$\mathbb{F} = \left\{ \pm 0.1 d_2 \cdots d_{t+1} \times 2^m \right\} \cup \{0\},$$

where $d_2, \ldots, d_{t+1} \in \{0, 1\}$ and $m \in [L, U]$.

- t is precision
- m is exponent

IEEE formats.

- **Single precision (32-bit):** $t = 23$, $L = -126$, $U = 127 = 2^7 - 1$ i.e. 1 bit for sign, 8 bits for exponent, 23 bits for precision.
- **Double precision (64-bit):** $t = 52$, $L = -1022$, $U = 1023 = 2^{10} - 1$.

1.1.3 Rounding

A real number $x \in \mathbb{R}$ is represented by

$$x \approx \mathrm{fl}(x) \in \mathbb{F}$$

Typical rounding choices:

- Round up / round down
- **Round to nearest (ties to even)** – IEEE 754 default

Thus an *inherent* error exists: $x \neq \mathrm{fl}(x)$.

1.2 Round off Error

Machine precision (unit roundoff).

$$\varepsilon_{\text{machine}} = 2^{-(t+1)}.$$

- Single: $2^{-24} \sim O(10^{-8})$.
- Double: $2^{-53} \sim O(10^{-16})$.

Example 1.2 (Round-off in trigonometric function). $\sin(5\pi) = 0$, yet a typical double-precision routine yields a value on the order of 10^{-16} (e.g. 6.12×10^{-16} using `julia`).

1.2.1 Cancellation in Sums

Adding (or subtracting) small numbers with big numbers would remove the precision in small numbers.

Example 1.3 (Small + Big goes wrong).

$$x = 0.23371258 \times 10^{-4}, \quad y = 0.33678429 \times 10^{2}, \quad z = -0.33677811 \times 10^{2}.$$

Exact: $x + y + z = 0.641371258 \times 10^{-3}$ (9 significant digits).

Add Small + Big first (worse):

$$\big(\mathrm{fl}(x) + \mathrm{fl}(y)\big) + \mathrm{fl}(z) = 0.64100000 \times 10^{-3}.$$

Add Big + Big first (better):

$$\mathrm{fl}(x) + \big(\mathrm{fl}(y) + \mathrm{fl}(z)\big) = 0.64137126 \times 10^{-3}.$$

1.2.2 Cancellation in Differences

The difference in large numbers is also unstable.

Example 1.4. Consider

$$\sqrt{1 + 10^{-16}} - 1.$$

In floating arithmetic,

$$\mathrm{fl}(\sqrt{1+10^{-16}}) - \mathrm{fl}(1) = 0$$

due to catastrophic cancellation.

Algebraic reformulation (stable).

$$\sqrt{1+10^{-16}} - 1 = \frac{10^{-8}}{\sqrt{1+10^{-16}}+1}.$$

Then in floating arithmetic,

$$\mathrm{fl}\left(\frac{10^{-8}}{\sqrt{1+10^{-16}}+1}\right) \approx 5 \times 10^{-9}.$$

Code demo (Julia-like).

- `sqrt(1+1e-16) - 1`
- `1e-8 / (sqrt(1+1e-16) + 1)`

1.2.3 Division by Small Numbers

Example 1.5.

$$\mathrm{fl}(10^{-15}) + 1 - 1 \approx 1.11 \times 10^{-15} \quad \text{(very small absolute error)},$$

$$\text{Exact:} \quad \frac{1+10^{-15}-1}{10^{-15}} = 1,$$

$$\text{Floating:} \quad \frac{\mathrm{fl}(10^{-15})+1-1}{\mathrm{fl}(10^{-15})} \approx 1.11 \quad \text{(huge relative error } \sim 11\%).$$

Code demo (Julia-like).

```
(1+1e-15) - 1
```

- `((1+1e-15)-1)/(1e-15)`

A Real-World Cautionary Tale Accumulated round-off can snowball over many steps. On June 4, 1996, the ESA *Ariane 5* rocket failed due to mishandling of floating-point conversions / round-off in software.

Summary

- Converting between binary and decimal
- Understanding round-off error and unit roundoff
- Typical pitfalls: (i) small+big sums, (ii) subtraction of close numbers, (iii) division by small numbers

References.

- Jeffrey R. Chasnov, *Numerical Methods*.
- Pingwen Zhang & Tiejun Li, *Numerical Analysis* (Chinese).

Chapter 2

Algorithms for Solving Non-Linear Equations

03/09/2025

Lecture 2: Bisection Method

Solve the roots (zeros) of a function. Given a function $f : \mathbb{R} \to \mathbb{R}$, find root(s) x^* satisfying

$$f(x^*) = 0.$$

2.1 Bisection Method

Recall the Intermediate Value Theorem (IVT):

Theorem 2.1 (Intermediate value theorem). Let $f \in \mathcal{C}^0[a, b]$ and $f(a)f(b) < 0$, then there exists $p \in (a, b)$ satisfying $f(p) = 0$.

This motivates the bisection method to find the root approximately.

Algorithm 1: Bisection method

Data: bounds $a, b \in \mathbb{R}$, $f \in \mathcal{C}^0[a,b]$ satisfying $f(a)f(b) < 0$, error ε
Result: approximate root p

```
a_1 ← a ;
b_1 ← b ;
i ← 1 ;
p_1 = (a_1 + b_1)/2 ;
while not condition do
    if f(a_i)f(p_i) > 0 then
        a_{i+1} ← p_i ;
        b_{i+1} ← b_i ;
    else
        a_{i+1} ← a_i ;
        b_{i+1} ← p_i ;
    i++ ;
    p_i = (a_i + b_i)/2 ;
p ← p_i ;
```

The condition can be

- $|p_i - p_{i-1}| < \varepsilon$: absolute change of iteration,
- $\frac{p_i - p_{i-1}}{p_i} < \varepsilon$: relative change of iteration,
- $|f(p_i)| < \varepsilon$: magnitude, or
- $i > i_{\max}$: maximum iteration.

Note that in other root-finding algorithms, we also employ one of these conditions.

Theorem 2.2 (Convergence speed of bisection method)**.** Assume the bisection method condition holds and such root p is unique. The bisection method generates a sequence $p_n \to p$ with

$$|p_n - p| \le \frac{b-a}{2^n}$$

Proof. Note that $p \in [a_n, b_n], p_n = \frac{a_n + b_n}{2}$ and $b_n - a_n = \frac{b-a}{2^{n-1}}$. □

Remark. *In other word, $p_n = p + O(2^{-n})$. Also, this bound is only the worst case, while the actual error is usually much smaller.*

This also means if we want to achieve $|p_n - p| < \varepsilon$, we need

$$n > \log_2 \frac{\varepsilon}{b-a}.$$

The convergence rate is

$$r = \inf\{r : \limsup_{k\to\infty} \frac{|p_{k+1} - p|}{|p_k - p|} < +\infty\}.$$

For bisection method, the convergence rate is linear since $|p - p_{n+1}| \le \frac{|p - p_n|}{2}$. This means the convergence is independent of the original range.

Advantage: Always converge.

Disadvantage: Does not work for higher dimension, modification required.

Lecture 3: Fixed Point Iteration

2.2 Fixed Point Iteration

Definition 2.3 (Fixed point). For a function $g : X \to X$, a point $p \in X$ is said to be a fixed point if $p = g(p)$.

Thus a root finding problem can be converted to a fixed point problem.

Theorem 2.4 (Existence of a fixed point). Let $g \in \mathcal{C}^0 : [a, b] \to [a, b]$, then there exists a fixed point in $[a, b]$.

Proof. Consider $h(x) = g(x) - x$. If $h(a) = 0$ or $h(b) = 0$ then we are done. Else, apply the intermediate value theorem. □

The uniqueness can be shown by the Banach fixed point theorem.

Theorem 2.5 (Banach fixed point theorem). Let (X, d) be a non-empty metric space and $f : X \to X$ be a contraction mapping i.e. there exists $0 \leq L < 1$ such that $|f(x) - f(y)| < L|x - y|$ for any $x, y \in X$, then there exists a unique fixed point for f.

Proof.

1. Show that f is continuous
2. For any $x_0 \in x$, define $x_n = f(x_{n-1})$. Show that (x_n) is a Cauchy sequence, thus converges.
3. Show that fixed point is unique by considering $|x^* - \bar{x}^*|$.

□

Since an interval of $\mathbb{R}$ is a metric space, this gives the fixed point method.

Algorithm 2: Fixed point method

Data: bounds $a, b \in \mathbb{R}$, $f :\in \mathcal{C}^0[a, b]$, initial guess $p_0 \in [a, b]$
Result: approximate root p

```
Rewrite f(x) = 0 ⇒ x = g(x), g ∈ C^0[a, b] ;
i ← 0 ;
while not condition do
    p_i ← g(p_i) ;
    i++ ;
p ← p_i ;
```

The condition can be

- $|p_i - p_{i-1}| < \varepsilon$
- $i > i_{\max}$.

In implementation, we would like to choose g to be a contraction mapping, then the error bound is

$$|p_n - p| \leq L^n |p_0 - p| \Rightarrow |p_n - p| = O(L^n).$$

A value of L can be easily found if g is differentiable.

Theorem 2.6 (Mean value theorem). Let g be a function continuous on $[a, b]$ and differentiable on (a, b), then there exists $c \in (a, b)$ such that

$$g'(c) = \frac{g(b) - g(a)}{b - a}.$$

In this case, $L = \sup_{x \in [a,b]} |g'(x)|$.

Also, note that the choice of g is crucial to whether, and how fast the algorithm converges.

Example 2.7. To solve $f(x) = x^3 + 4x^2 - 10 = 0$, we have these choices for fixed point problem $x =$

- $g_1(x) = x - x^3 - 4x^2 + 10$;
- $g_2(x) = \sqrt{\frac{10}{x} - 4x}$;
- $g_3(x) = \frac{\sqrt{10 - x^3}}{2}$;
- $g_4(x) = \sqrt{\frac{10}{4 + x}}$;
- $g_5(x) = x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}$.

These choices give different results

- g_1: diverges;
- g_2: diverges and goes out of domain;
- g_3: 10 s.f. precision in 30 iterations;
- g_4: 10 s.f. in 15 iterations;
- g_5: 10 s.f. in 4 iterations.

Advantage: very fast if contraction map chosen correctly, easily implemented on a calculator

Disadvantage: choosing a wrong contraction map leads to divergence

15/09/25

Lecture 4: Newton Method

2.3 Newton Method

Consider $f \in \mathcal{C}^2[a, b]$ with a root $f(p) = 0$, Taylor expansion gives

$$f(p) = f(p_0) + (p - p_0) f'(p_0) + \frac{(p - p_0)^2}{2} f''(\xi),$$

where ξ is between p, p_0. Hence, we get

$$p = p_0 - \frac{f(p_0)}{f'(p_0)} - \frac{(p - p_0)^2}{2} \frac{f''(\xi)}{f'(p_0)} \approx p_0 - \frac{f(p_0)}{f'(p_0)}.$$

This gives the Newton's method.

Algorithm 3: Newton's method

Data: bounds $a, b \in \mathbb{R}$, f differentiable in (a, b), initial guess $p_0 \in [a, b]$
Result: approximate root p

```
i ← 0 ;
while not condition do
    if f'(p_i) = 0 then
        Raise division-by-zero exception. ;
    else
        p_{i+1} = p_i − f(p_i)/f'(p_i) ;
    i++ ;
p ← p_i ;
```

Geometrically, this gives the intersection between the tangent line at p_i and the x-axis.

Theorem 2.8 (Quadratic convergence rate). Assume $f \in \mathcal{C}^2[a, b]$ satisfies $f(p) = 0$ and $f'(p) \neq 0$, then for a closed neighborhood of p, the Newton's method converges in quadratic rate. In other words, there exists $\delta > 0$, such that for any $p_0 \in [p_0 - \delta, p_0 + \delta]$, the sequence

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$$

converges to p and

$$\limsup_{n\to\infty} \left| \frac{p - p_n}{(p - p_{n-1})^2} \right| < +\infty.$$

Proof.

1. Shows that the Newton's method is equivalent to a fixed point iteration
2. Show that there is a contraction mapping.
3. Taylor expansion around p_n to estimate error.

□

To prevent overshooting, we may use adaptive step size instead

$$p_{i+1} = p_i - \lambda_i \frac{f(p_i)}{f'(p_i)}.$$

This slows down the error decay rate but relaxes the requirement for initial guess.

17/09/25

Lecture 5: Secant Method and comparison of different methods

2.4 Secant Method

Motivation: When computation of derivative is unfeasible, approximate derivative by

$$f'(p_n) \approx \frac{f(p_n) - f(p_{n-1})}{p_n - p_{n-1}},$$

as both terms are same in the limit $n \to \infty$.

Algorithm 4: Secant method

Data: function f, initial guesses p_0, p_1
Result: approximate root p

```
i ← 1 ;
while not condition do
    p_{i+1} = p_i − f(p_i) (p_i − p_{i−1}) / (f(p_i) − f(p_{i−1})) ;
    i++ ;
```

Theorem 2.9. If $f \in \mathcal{C}^2[a,b]$ and there exists $p \in (a,b)$ such that $f(p) = 0$ and $f'(p) \neq 0$, then within an open neighborhood of p, the sequence generated by the secant method converges to p and convergence rate is $\phi = \frac{1+\sqrt{5}}{2}$.

Advantages: superlinear convergence, no need for derivatives
Disadvantages: slower than Newton's method, requires good initial guesses

The method of false position combines bisection method and secant method for both robustness and speed. Bisection is stable but slow (linear), secant is fast (superlinear) but unstable.

Algorithm 5: Method of False Position

Data: initial guesses p_0, p_1, function $f \in \mathcal{C}^0[p_0, p_1]$ with $f(p_0)f(p_1) < 0$
Result: approximate root p

```
i ← 1 ;
a_1 ← p_0 ;
b_1 ← p_1 while not condition do
    p_{i+1} = b_i − f(b_i) (b_i − a_i) / (f(b_i) − f(a_i)) ;
    if f(p_{i+1}) f(b_i) < 0 then
        b_{i+1} ← b_i ;
        a_{i+1} ← p_{i+1} ;
    else
        b_{i+1} ← p_{i+1} ;
        a_{i+1} ← a_i ;
    i++
```

i.e. Now we find the secant from points with different signs in function values.

24/09/25

Lecture 6: Extensions to High Dimensional Nonlinear Systems and Optimization

2.5 High Dimensional Generalization

2.5.1 Fixed Point Iteration

Definition 2.10 (Normed vector space). $(V, \|\cdot\|)$ is said to be a normed vector space if V is a vector space and $\|\cdot\| : V \to \mathbb{R}$ is the norm function such that $\forall \mathbf{x}, \mathbf{y} \in V, c \in \mathbb{R}$

- (Positive definite) $\|\mathbf{x}\| \geq 0, = 0 \iff \mathbf{x} = \mathbf{0}$,

- (Absolute homogeneity) $\|c\mathbf{x}\| = |c|\|\mathbf{x}\|$,
- (Triangle inequality $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.

Example 2.11. For the Euclidean space $\mathbb{R}^d$, the 2-norm is

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \cdots + x_d^2}.$$

While the 2-norm is commonly used in $\mathbb{R}^d$ due to rotational invariance, other norms share the same topological property.

Proposition 2.12. All norms in $\mathbb{R}^d$ are equivalent i.e. for any two norms $\|\cdot\|, \|\cdot\|'$, there exists $C > 0$ such that for any $\mathbf{x} \in \mathbb{R}^d$

$$\frac{\|\mathbf{x}\|}{C} \leq \|\mathbf{x}\|' \leq C\|\mathbf{x}\|.$$

Also, note that

Proposition 2.13. Normed vector space is a metric space with distance function $d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|$.

Hence, we can apply the Banach fixed point theorem to guarantee a fixed point. The algorithm is same as Algorithm 2, but replacing the interval with a subset of $\mathbb{R}^d$.

Note that in fixed point iteration, we only used information for $\mathbf{p}_i$ to build $\mathbf{p}_{i+1}$. If extra m entries of history information i.e. $\mathbf{p}_n, \cdots, \mathbf{p}_{n-m}$, we can minimize the residual by finding a convex combination.

Algorithm 6: Anderson acceleration

Data: subset $D \subset \mathbb{R}^d$, function $\mathbf{F} : D \to D$, initial guess $\mathbf{p}_0 \in D$, number of historic entries $m \in \mathbb{N}$
Result: approximate root $\mathbf{p}$

Rewrite $\mathbf{F}(\mathbf{x}) = \mathbf{0} \iff \mathbf{x} = \mathbf{G}(\mathbf{x})$;
$i \leftarrow 0$;
$\mathbf{r}_0 = \mathbf{G}(\mathbf{p}_0) - \mathbf{p}_0$;
while *not condition* **do**
  $m_i \leftarrow \min(m, i)$;
  $\hat{\boldsymbol{\alpha}} \leftarrow \operatorname{argmin}_{\boldsymbol{\alpha}} \|\alpha_0 \mathbf{r}_0 + \cdots + \alpha_{m_i}\mathbf{r}_{m_i}\|_2 : \sum_{k=0}^{m_i} \alpha_k = 1$;
  $\mathbf{p}_{i+1} \leftarrow \sum_{k=0}^{m_i} \hat{\alpha_k} \mathbf{G}(p_{i-m_i+k})$;
  $\mathbf{r}_{i+1} = \mathbf{G}(\mathbf{p}_{i+1}) - \mathbf{p}_{i+1}$;
  $i++$;

This algorithm has a superlinear convergence compared to the linear convergence

2.5.2 Newton Method

Definition 2.14 (Jacobian matrix). The $d \times d$ Jacobian matrix $\mathbf{J}(\mathbf{x})$ of a function $\mathbf{f}(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$ is defined by

$$\mathbf{J} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_d}{\partial x_1} & \cdots & \frac{\partial F_d}{\partial x_d} \end{pmatrix}$$

i.e. $J_{ij} = \frac{\partial J_i}{\partial x_j}$.

The Jacobian matrix generalizes the derivative of a single-variable function. The algorithm is similar to Algorithm 3, while replacing the derivative with $\mathbf{J}$.

Similar with the 1D case, the convergence speed is quadratic, and a good initial guess is required. However, the computation of $\mathbf{J}^{-1}$ can be expensive. This motivates the quasi-Newton method to approximate $\mathbf{J}_i^{-1}$ by cheap computations (JFNK), which also benefit from the super-linear convergence speed.

2.6 Optimization

Aim: find $\mathbf{p} \in \mathbb{R}^d$ minimizing $f(\mathbf{x})$.

If f is differentiable, then at the local minima, $\nabla f(\mathbf{x}) = 0$

Algorithm 7: Gradient descent

Data: Differentiable $f : \mathbb{R}^d \to \mathbb{R}$, learning rate $\alpha > 0$, initial point $\mathbf{x}_0 \in \mathbb{R}^d$
Result: approximate argmin $\mathbf{x} : \nabla f(\mathbf{x}) \approx \mathbf{0}$

```
i ← 0 ;
while not condition do
    x_{i+1} = x_i − α∇f(x_k) ;
    i++;
x ← x_i;
```

The condition is $i > i_{\max}$ or $\|\nabla f(\mathbf{x}_i)\| \leq \varepsilon$.

The problem can also be seen as solving the fixed point problem $\mathbf{x} = \mathbf{x} - \alpha \nabla f(\mathbf{x})$.

Advantage: intuitive, simple, guaranteed to obtain a solution for sufficiently small α

Disadvantage: overshooting α may lead to unstable solution, maxima or saddle point.

Alternatively, we can use the Newton method. Before so, we would like to generalize the second derivative by the Hessian matrix.

Definition 2.15 (Hessian matrix). The Hessian matrix $\mathbf{H}(\mathbf{x})$ of a function $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ is

$$\mathbf{H} = \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d x_1} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix}.$$

Algorithm 8: Newton method for optimization

Data: subset $D \subset \mathbb{R}^d$, $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R} \in \mathcal{C}^2$, initial guess $\mathbf{x}_0$
Result: approximate minimizer $\mathbf{x} : \nabla f(\mathbf{x}) \approx \mathbf{0}$

```
i ← 0 ;
while not condition do
    x_{i+1} = x_i − H_i^{-1} ∇f(x_i) ;
    i++ ;
```

Although the convergence is quadratic, computing the Hessian matrix can be expensive.

Chapter 3

Numerical Interpolation

2025/09/29

Lecture 7: Lagrange Interpolation

Given a set of data points related by some unknown function we want to approximate the function values at other points via interpolation. Let $\{(x_i, y_i)\}$ be a set of data points satisfying $y_i = f(x_i)$, we would like to find

$$f(x), x \neq x_i.$$

One popular choice is polynomials

$$P_n(x) = a_n x^n + \cdots + a_0$$

because

- continuous;
- easy derivatives and integrals calculation;
- uniformly approximate any continuous functions.

Theorem 3.1 (Weierstrass approximation theorem)**.** Let $f \in \mathcal{C}^0[a, b]$, then for any $\varepsilon > 0$, there exists a polynomial $p(x)$ so that

$$\forall x \in [a, b], |f(x) - p(x)| < \varepsilon.$$

3.1 Lagrange Interpolation

With the well convergence guaranteed, the next step is to find a "good" polynomial. The first insight would be Taylor expansion

$$f(x) \approx \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

However, this approximation is bad because only relies on local information.

Example 3.2. $f(x) = e^x$, the error becomes worse further away from $x_0 = 0$.

Example 3.3. Even worse, consider $f(x) = \frac{1}{x} \approx \sum_{k=0}^{n} (-1)^k (x-1)^k$, the value diverges for $|x-1| \geq 1$.

Also, Taylor expansion requires derivative information, which requires f to be smooth.
This comes to the idea of Lagrange polynomial.

Algorithm 9: Lagrange interpolation

Data: interpolation points $(x_i, y_i)_{i=0}^n$ satisfying $y_i = f(x_i)$
Result: polynomial L satisfying $L(x_i) = y_i$

1 Construct the Lagrange basis L_j satisfying

$$\begin{aligned} L_j(x_i) &= \delta_{jk} \\ &= \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_j-x_0)\cdots(x_j-x_{k-1})(x_j-x_{k+1})\cdots(x_j-x_n)} \\ &= \prod_{i=0,\neq j}^{n} \frac{x-x_i}{x_j-x_i} \end{aligned}$$

;

2

$$\begin{aligned} L(x) &= y_0 L_0(x) + \cdots + y_n L_n(x) \\ &= \sum_{i=0}^{n} y_j L_j(x) \end{aligned}$$

The error bound can be found by the Lagrange remainder formula

Theorem 3.4 (Error bound of Lagrange interpolation). Suppose $x_i \in [a,b]$ and $f \in \mathcal{C}^{n+1}[a,b]$, then

$$f(x) = L(x) + \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x-x_0)\cdots(x-x_n).$$

Hence, the error is

$$|f(x) - L(x)| \leq \max_{[a,b]} \left| \frac{f^{(n+1)}(x)}{(n+1)!}(x-x_0)\cdots(x-x_n) \right|.$$

This means the Lagrangian interpolation captures polynomials with degree at most n.
Before starting the proof, we need a lemma.

Lemma 3.5 (Generalized Rolle's theorem). Let $f \in \mathcal{C}^0[a,b]$ be n-times differentiable on (a,b). If $f(x) = 0$

at $n+1$ distinct points $x_0 \le \cdots \le x_n, x_i \in [a,b]$, then there exists a number $c \in [x_0, x_n] \subset (a,b)$ with

$$f^{(n)}(c) = 0.$$

Proof. For $n = 1$, if f is not a constant function, then the maximum or minimum exists in some $c \in (a,b)$. Then $f'(c) = 0$ considering limits on both sides. For $n > 1$, there are n points with $f' = 0$, so continue by induction. □

Now we give a proof for the error bound of Lagrange interpolation.

Proof. For $x = x_i$, the relation is trivial since one of $x - x_i = 0$. For $x \neq x_i$, define

$$g(t) = f(t) - L(t) - (f(x) - L(x)) \prod_{i=0}^{n} \frac{t - x_i}{x - x_i}.$$

Note that $g \in \mathcal{C}^{n+1}[a,b]$ and $g(x_i) = 0$. So by the generalized Rolle's theorem, there exists $\xi \in (a,b)$ satisfying $g^{(n+1)}(\xi) = 0$. The equality can be shown considering $L^{(n+1)} = 0$ and $\frac{d^{n+1}}{dt^{n+1}} \prod_{i=0}^{n} \frac{t-x_i}{x-x_i} = \frac{1}{\prod_{i=0}^{n}(x-x_i)}$. □

2025/10/06

Lecture 8: Divided Difference and Newton Interpolation

3.2 Newton Interpolation

Motivation: Suppose we have the Lagrangian interpolation for $\{(x_i, y_i)\}_{i=0}^{n}$. Now we have a new data point (x_{n+1}, y_{n+1}), we want to reuse the information instead of reconstructing the interpolation.

Let $L_n(x)$ be the n-th order Lagrange polynomial satisfying $L_n(x_i) = f(x_i)$. Alternatively, it can be expressed as

$$L_n(x) = a_0 + a_1(x - x_0) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1}).$$

Note that

$$\begin{cases} f(x_0) & = a_0 \\ f(x_1) & = a_0 + a_1(x_1 - x_0) \Rightarrow a_1 = \frac{f(x_1) - f(x_0)}{(x_1 - x_0)}. \end{cases}$$

This motivates the divided difference

Definition 3.6 (Divided difference)**.** The divided difference of function f is

$$f[x_0, \cdots, x_k] = \begin{cases} \frac{f[x_1, \cdots, x_k] - f[x_0, \cdots, x_{k-1}]}{x_k - x_0}, & k \neq 0 \\ f(x_0), & k = 0 \end{cases}.$$

In calculation, we can write down the value of $f(x_i)$ in each column and evaluate column-by-column. We can express the coefficients in terms of divided difference.

Proposition 3.7. $a_k = f[x_0, \cdots, x_k]$

Hence, this method is known as Newton interpolation, which give the same polynomial.

Algorithm 10: Newton Interpolation

Data: interpolation points $(x_i, y_i)_{i=0}^n$ satisfying $y_i = f(x_i)$
Result: Newton interpolation polynomials $P_k, 0 \le k \le n$ satisfying $L(x_i) = y_i$ for $0 \le k \le i$

$P_0(x) \leftarrow f[x_0]$;
for $k \leftarrow 1$**to** n **do**
  $P_k(x) = P_{k-1}(x) + f[x_0, \cdots, x_k](x - x_0) \cdots (x - x_k)$;

Theorem 3.8 (Error of Newton Interpolation). Let $f : [a, b] \to \mathbb{R}$ and $P_n(x)$ be the Newton interpolation with $x_0, \cdots, x_n$. Then for $x \in [a, b]$, we have

$$f(x) - P_n(x) f[x, x_0, \cdots, x_n] \prod_{i=0}^{n} (x - x_i).$$

In other words, the error is

$$|f(x) - P_n(x)| \le \sup_{x \in [a,b]} |f[x, x_0, \cdots, x_n]| \prod_{i=0}^{n} |x - x_i|.$$

Proof. Expand $f[x, x_0, \cdots, x_n]$ and divided differences containing x, and then use the definition of Newton interpolation. □

2025/10/08

Lecture 9: Piecewise Interpolation

3.3 Piecewise Interpolation

Problem of Lagrange interpolation: consider Runge function $R(x) = \frac{1}{1+x^2}$, the $R^{(n)}(x)/n!$ is 1 at $x = 0$, and $\prod_{i=0}^{n}(x - x_i)$ grows large.

As the distance between interpolation points are small locally, we can construct a piecewise interpolation with just local information.

Algorithm 11: Piecewise linear interpolation

Data: Interpolation points $\{(x_i, y_i = f(x_i))\}_{i=0}^n$, WLOG $x_0 < \cdots < x_n$
Result: Piecewise linear interpolation$P_n(x) : [x_0, x_n] \to \mathbb{R}$

Construct piecewise linear basis $I_i(x_j) = \delta_{ij}$, which are triangular spikes satisfying $I_i(x_{i-1}) = I_i(x_{i+1}) = 0, I_i(x_i) = 1.$;

$$P_n(x) = \sum_{i=0}^{n} y_i I_i(x)$$

;

Theorem 3.9 (Error of piecewise linear interpolation). The error is

$$|P_n(x) - f(x)| \leq \frac{\max_{1\leq i\leq n} |x_i - x_{i-1}|^2}{8} \max_{[x_0,x_n]} |f''|.$$

Proof. Fix x, if $x = x_i$, the inequality is trivial. For $x \neq x_i$, choose interval $x \in (x_i, x_{i+1})$ and consider

$$w(t) = f(t) - P_n(t) - (f(x) - P_n(x))\frac{(t - x_i)(t - x_{i+1})}{(x - x_i)(x - x_{i+1})}.$$

Note that $w(x_i) = w(x_{i+1}) = w(x) = 0$. By generalized Rolle's theorem, there exists $\xi \in (x_i, x_{i+1})$ so that $w''(\xi) = 0$. Arranging back the terms give

$$f(x) - P_n(x) = \frac{f''(\xi)}{2}(x - x_i)(x - x_{i+1}).$$

Using $|x - x_i||x - x_{i+1}| \leq \frac{(|x_{i+1}-x|+|x-x_i|)^2}{4} = \frac{(x_{i+1}-x_i)^2}{4}$ gives

$$|P_n(x) - f(x)| \leq \frac{\max_{1\leq i\leq n} |x_i - x_{i-1}|^2}{8} \max_{[x_0,x_n]} |f''|. \quad \square$$

$\square$

Problem: undifferentiable kninks at the knots x_i.
Solution: k-th order spline, $S_k(x) \in \mathcal{C}^{k-1}(x_0, x_n)$ satisfying

- piecewise polynomial with degree $\leq k$,
- $S_k(x_i) = y_i$,

Usually they take $k = 3$ i.e. cubic spline, which is defined to be

$$S_{3,i}(x) = \frac{m_i(x_{i+1} - x_i)^3}{6h_i} + \frac{m_{i+1}(x - x_i)^3}{6h_i} + (y_i - \frac{m_i h_i^2}{6})\frac{x_{i+1} - x_i}{h_i} + (y_{i+1} - \frac{m_{i+1}h_i^2}{6})\frac{x - x_i}{h_i}.$$

It satisfies

- $S''_{3,i}(x_i^+) = m_i$,
- $S_{3,i}(x_{i+1}^-) = m_{i+1}$,
- $S_{3,i}(x_i) = y_i$,
- $S_{3,i}(x_{i+1}) = y_{i+1}$.

Continuity of S'_3 is ensured by solving a linear system consists of knots.
Boundary condition

- Natural: $S''(x_0) = S''(x_n) = 0 \Rightarrow m_0 = m_N = 0$
- Clamped: $S'(x_0) = \alpha, S''(x_n) = \beta$

Chapter 4

Numerical Differentiation and Integration

2025/10/13

Lecture 10: Numerical Differentiation

4.1 Numerical Differentiation

Given data points $\{(x_i, y_i = f(x_i)\}, x_i \approx x$, we would like to approximate $f'(x)$.

The derivative of f is given by

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

Simply, this motivates the forward difference

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Similarly, we have the backward difference

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

and the central difference

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Proposition 4.1 (Error bounds of simple finite difference methods). The forward and backward difference methods have $O(h)$ error, while the central difference method have $O(h^2)$ error.

Proof. Expand $f(x-h), f(x), f(x+h)$ by Taylor series up to appropriate order, so only the derivative and residue remain. □

Obviously, we have

Algorithm 12: Finite difference approximation for derivatives.

Data: Data points $\{(x_i(x,h), f(x_i))\}_{i=1}^n$
Result: Approximate derivative $f'(x)$
1 Expand each of $f(x_i)$ around x up to nth order by Lagrange remainder theorem ;
2 Match each coefficient in lowest orders of h, solve the system of linear equation and find the corresponding coefficients.

Proposition 4.2. The error from approximation points is $O(h^{k-1})$.

Equivalently, we can consider Lagrange interpolation on the data points and then take the derivative of the polynomial, but it is too complicated.

Example 4.3. Given the values at $x, x+h, x+2h$, we let $f'(x) \approx \frac{c_0 f(x)+c_1 f(x+h)+c_2 f(x+2h)}{h}$. Taylor approximation gives

$$\begin{cases} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f^{(3)}(\xi_1) \\ f(x+2h) &= f(x) + 2hf'(x) + 2h^2 f''(x) + \frac{4h^3}{3} f^{(3)}(\xi_2) \end{cases}.$$

Comparing coefficients of $1, h, h^2$ gives

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & \frac{1}{2} \\ 1 & 2 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \Rightarrow c_0 = -3, c_1 = 4, c_2 = 2$$

The error term is $O(h^2)$

To compute higher-order derivatives, we can use the same technique, except the target coefficients (on the RHS) are different.

2025/10/20

Lecture 11: Numerical Integration

4.2 Integration

Approximate $\int_a^b f(x)dx \approx \sum_{i=0}^n a_i f(x_i)$.

4.2.1 Newton-Cotes quadrature

Given the data points $\{x_i, f(x_i)\}_{i=0}^n \subset [a,b]$, construct Lagrange interpolating polynomial $P_n(x)$ and approximate $\int_a^b f(x)dx \approx \int_a^b P_n(x)dx$. The error is thus

$$e = \frac{1}{(n+1)!} \int_a^b \prod_{i=0}^n (x - x_i) f^{(n+1)}(\xi) dx.$$

0th-order approximation: approximate function by a constant
1st-order approximation: approximate by an affine function

Rule	x_i	$a_i/(b-a)$	**Error bound**
Left	a	1	$\frac{M_1(b-a)^2}{2}$
Right	b	1	$\frac{M_1(b-a)^2}{2}$
Mid-point	$\frac{a+b}{2}$	1	$\frac{M_2(b-a)^3}{24}$
Trapezoidal	a, b	1/2, 1/2	$\frac{M_2(b-a)^3}{12}$

Table 4.1: Low order approximation rules for integrals.

Note that $M_k := \sup_{[a,b]} f^{(k)}$.

While the error for 0-th order approximation can be found by Taylor expansion, the error for trapezoidal rule can be approximated by the integral mean value theorem.

Theorem 4.4 (Integral mean value theorem). Let $f \in \mathcal{C}^0[a,b], g \in \mathcal{R}[a,b]$ and g does not change sign on $[a,b]$, then there exists $c \in (a,b)$ satisfying

$$\int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx.$$

If we approximate the function as a polynomial at points $a, \frac{a+b}{2}, b$, we get the Simpson's rule

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

with error

$$\frac{M_4(b-a)^5}{180}.$$

However, when $(b-a)$ becomes large, or the function have large derivatives. the error will be large. From piecewise interpolation, we have composite quadrature: applying a quadrature in each sub-intervals $[x_{i-1}, x_i]$ with length $h = \frac{n}{b-a}$.

Rule	x_i^*	a_i/h	**Error bound**
Mid-point	$\frac{x_{i-1}+x_i}{2}, i > 0$	1	$\frac{M_2(b-a)^3}{24n^2}$
Trapezoidal	$x_0, \cdots, x_n$	$1/2, 1, \cdots, 1, 1/2$	$\frac{M_2(b-a)^3}{12n^2}$
Simpson's	$x_0, \cdots, x_n$	$1/6, 4/6, 2/6, 4/6, 2/6, \cdots, 4/6, 1/6$	$\frac{M_4(b-a)^5}{180n^4}$

Table 4.2: Composite quadrature rules for integration

4.2.2 Gauss Quadrature

Alternatively, we can see the accuracy from other perspective.

Definition 4.5 (Degrees of accuracy of quadrature formula). A quadrature formula is *n-th order accurate* iff the error is zero for all polynomials degree within n, but not some of degree $n+1$.

Approximating $\int_a^b f(x)dx \approx \sum_{i=0}^n \omega_i f(x_i)$, we have $2(n+1)$ unknowns, so the quadrature $(2n+1)$-order accurate. The equations are

$$\begin{cases} \omega_0 + \cdots + \omega_n & = b-a \\ \omega_0 x_0 + \cdots + \omega_n x_n & = \frac{b^2-a^2}{2} \\ & \vdots \\ \omega_0 x_0^{2n+1} + \cdots + \omega_n x_n^{2n+1} & = \frac{b^{2n+2}-a^{2n+2}}{(2n+1)!} \end{cases}$$

Problem: unbounded ends

Solution: weighted Gauss quadrature $\int_a^b f(x)w(x)dx \approx \sum_{i=0}^n \omega_i f(x_i)$, so that both sides agree for polynomials with degree within n.

Examples

- Gauss-Chebyshev: $\int_{-1}^1 \frac{f(x)}{\sqrt{1-x^2}}dx$
- Gauss-Laguerre: $\int_0^{+\infty} e^{-x} f(x)dx$
- Gauss-Hermite: $\int_{-\infty}^{+\infty} e^{-x^2} f(x)dx$

2025/10/22

Lecture 12: Truncation Error & Stability of Numerical ODE

4.3 ODE

Aim: Solve the initial value problem (IVP)

$$\begin{cases} \frac{dy}{dt} & = f(t,y), t \in [0,T] \\ y(0) & = y_0 \end{cases}$$

on a mesh $0 = t_0 \le \cdots \le t_N = T$.

Denote $y_k \approx y(t_k)$ as the approximate solution, we initialize $y_0 : +y(t_0)$ first then update y_{k+1} from $y_0, \cdots, y_k$.

4.3.1 Euler Methods

Using a uniform mesh $t_n = nh, h = T/N$, we approximate the derivative $y'(t_n)$ to the 1st order by

$$y'(t_n) \approx \begin{cases} \frac{y_{n+1}-y_n}{h} & \text{Forward} \\ \frac{y_n - y_{n-1}}{h} & \text{Backward} \end{cases}.$$

Then we update each step by

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n) && \text{Forward;} \\ y_{n+1} - hf(t_{n+1}, y_{n+1}) &= y_n && \text{Backward.} \end{aligned}$$

Equivalently, they can be written as approximation schemes. For example, the forward Euler method is given by

$$L_h(y(t_n)) = \frac{y_{n+1} - y_n}{h} - f(t_n, y_n)$$

satisfying $L_h(y_h(t_n)) = 0$

Definition 4.6 (Local truncation error)**.** The local truncation error is $L_h(y(t_n))$, where $y(t_n)$ is the exact solution. The method is said to be p-th order accurate if $T_n = O(h^p)$.

Proposition 4.7. The forward Euler method is 1st order accurate.

Proof. Expand $y(t_{n+1})$ to the second order using Lagrange remainder theorem. We have $T_n \leq \frac{1}{2}\sup_{(0,T)}|y''(t)|h$. □

While the local truncation error focuses on each step, the global error focuses on the whole interval, given by $y_N - y(t_N)$.

Proposition 4.8. Suppose $f(t, y)$ is Lipschitz continuous in y i.e. there exists $L > 0$ such that $|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|$. Suppose there is an unique solution $y(t)$ to the IVP. Let $M = \sup_{[a,b]} y''(t)$. The global error of the forward Euler method is $O(h)$.

Proof. The global error is

$$\begin{aligned}
|e_N| &\leq |e_{N-1}| + Lh|y(t_{N-1}) - y_{N-1}| + \frac{h^2}{M} \\
&\leq (1 + Lh)|e_{N-1}| + \frac{h^2}{M} \\
&\leq \cdots \\
&\leq (1 + Lh)^N|e_0| + Mh\frac{(1 + Lh)^{n+1} - 1}{2L} \\
&\leq \exp(LT)|e_0| + \frac{Mh}{2L}(\exp(LT) - 1)
\end{aligned}$$

With exact initial condition, $e_0 = 0$, so the global error is $O(h)$. □ □

Next, we would like to study the stability of the method. This determines whether the method converges.

Definition 4.9. Absolute Stability