Elementary Computational Mathematics

Yimin Zhong

December 3, 2023

Contents

Chapter 1

Floating Point Arithmetic

In this chapter, we will introduce some basics of the real number system for modern computers and discuss the arithmetic operations of the number system.

1.1 Representation of Real Numbers

Any *nonzero* real number $x \in \mathbb{R}$ can be accurately represented with an infinite sequence of digits. This can be understood as the consequence that rational numbers are dense on any interval. Therefore, with the binary representation, we can write

$$x = \pm(0.d_1 d_2 d_3 \dots d_{t-1} d_t d_{t+1} \dots) \times 2^e,$$

where e is an integer exponent and $d_1 = 1$, the other binary digits $d_i \in \{0, 1\}$. The mantissa part

$$0.d_1 d_2 d_3 \dots = \frac{d_1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots .$$

Remark 1.1.1. *To guarantee the uniqueness of the above representation, we need a further assumption that there exists an infinite subset $S \subset \mathbb{N}$ that $d_j \neq 1$ for $j \in S$. For example, under binary representation*

$$0.111 \dots = (0.1) \times 2^1,$$

then we will take the latter representation.

1.2 Floating Point Numbers

The floating point numbers generally refer to a set of real numbers with *finite* mantissa length. More precisely, we consider the set of real numbers $\mathbb{F} = \mathbb{F}(t, e_{\text{min}}, e_{\text{max}}) \subset \mathbb{R}$ that

$$\mathbb{F} := \{x \in \mathbb{R} \mid x = \pm(0.d_1 d_2 d_3 \dots d_{t-1} d_t) \times 2^e, d_1 = 1, e_{\text{min}} \le e \le e_{\text{max}}\} \cup \{0\}.$$

It can be seen that there are only finite numbers in $\mathbb{F}$ with the smallest positive element $x_{\text{min}} = 2^{e_{\text{min}}-1}$ and the largest element $x_{\text{max}} = (1 - 2^{-t}) \times 2^{e_{\text{max}}}$. Therefore

$$\mathbb{F} \subset \overline{\mathbb{F}} := \{x \in \mathbb{R} \mid x_{\text{min}} \le |x| \le x_{\text{max}}\} \cup \{0\}.$$

Remark 1.2.1. *The elements in $\mathbb{F}$ are called normalized. If we allow $d_1 = 0$ in the definition of $\mathbb{F}$, then the numbers in the set are called denormalized.*

Theorem 1.2.2 (Distribution of floating numbers)**.** *For any $e_{\text{min}} \leq e \leq e_{\text{max}}$, the distribution of the floating point number system $\mathbb{F}$ on interval $[2^{e-1}, 2^e]$ is equidistant with distances of length 2^{e-t}.*

Proof. For any $x \in \mathbb{F} \cap [2^{e-1}, 2^e]$ it can be represented by

$$x = (0.d_1 d_2 \dots d_t) \times 2^e$$

where $d_1 = 1$. The mantissa is equidistantly distributed with distance 2^{-t}, therefore the floating point numbers are equidistantly distributed with distances of length 2^{e-t}. □

To understand the approximation to real numbers by the floating point number system $\mathbb{F}$, it is important to consider the maximal relative distance between the numbers in $\overline{\mathbb{F}}$ and their respective closest element in $\mathbb{F}$, which is the following quantity:

$$\max_{x \in \overline{\mathbb{F}}, x \neq 0} \min_{z \in \mathbb{F}} \frac{|z - x|}{|x|}.$$

The following holds:

Theorem 1.2.3 (Machine precision)**.**

$$\max_{x \in \overline{\mathbb{F}}, x \neq 0} \min_{z \in \mathbb{F}} \frac{|z - x|}{|x|} \leq 2^{-t}.$$

The number $\mathrm{u} := 2^{-t}$ is also called rounding unit or machine precision.

Proof. Without loss of generality, we only need to consider the positive numbers in $\overline{\mathbb{F}}$, then one can represent any nonzero $x \in [x_{\text{min}}, x_{\text{max}}]$ by

$$x = (0.d_1 d_2 \dots d_t \dots) \times 2^e \in [2^{e-1}, 2^e].$$

Since the floating point numbers are equidistantly distributed on $[2^{e-1}, 2^e]$ from Theorem 1.2.2, one can find $z^* \in \mathbb{F}$ such that

$$|z^* - x| \leq \frac{1}{2} 2^{e-t},$$

therefore

$$\frac{|z^* - x|}{|x|} \leq \frac{1}{2} 2^{e-t} \frac{1}{2^{e-1}} = 2^{-t}.$$

□

Remark 1.2.4. *On modern computers, the following two floating point number systems*

$$\mathbb{F}_{32} := \mathbb{F}(24, -125, 128), \quad \mathbb{F}_{64} := \mathbb{F}(53, -1021, 1024)$$

are supported, they are often called single precision and double precision, respectively.

1.3 Rounding

The rounding operation fl is to map any real numbers of $\overline{\mathbb{F}}$ into the floating point number system $\mathbb{F}$ with the smallest error. Such rounding operation can be written out explicitly, let $= \pm(0.d_1d_2\dots d_t d_{t+1}\dots) \times 2^e$, then

$$\text{fl}(x) = \begin{cases} \pm(0.d_1d_2\dots d_t) \times 2^e & \text{if } d_{t+1} = 0, \\ \pm(0.d_1d_2\dots d_t + 2^{-t}) \times 2^e & \text{if } d_{t+1} = 1. \end{cases}$$

It is clear that rounding fl is monotone and idempotent, which means

- $x \le y \Rightarrow \text{fl}(x) \le \text{fl}(y)$.
- $\text{fl}(z) = z$ if $z \in \mathbb{F}$.

Theorem 1.3.1 (Rounding error). *For any $x \in \overline{\mathbb{F}}$, $|\text{fl}(x) - x| = \min_{z\in\mathbb{F}} |z - x|$. If $x \neq 0$, then*

$$\frac{|\text{fl}(x) - x|}{|x|} \le \text{u} = 2^{-t}.$$

Proof. The special case that $x = 0$ is trivial, we only consider $x \in [x_{\text{min}}, x_{\text{max}}]$, it can be seen that

$$|\text{fl}(x) - x| = |(0.d_1d_2\dots \tilde{d_t}) - (0.d_1d_2\dots d_t d_{t+1}\dots)| \times 2^e \le 2^{-(t+1)} \times 2^e,$$

where $\tilde{d_t}$ is the rounding bit, therefore

$$\frac{|\text{fl}(x) - x|}{|x|} \le \frac{2^{e-(t+1)}}{2^{e-1}} = 2^{-t}.$$

□

Corollary 1.3.2. *For any $x \in \overline{\mathbb{F}}$, $\text{fl}(x) = x(1 + \delta)$ with $|\delta| \le \text{u}$.*

1.4 Arithmetic Operations

Let $\mathbb{F} = \mathbb{F}(t, e_{\text{min}}, e_{\text{max}})$ be a given floating point number system and we consider the basic binary operation $\circ \in \{+, -, *, /\}$ on $\mathbb{F}$, to represent the outcome in $\mathbb{F}$, a straightforward realization is to define the binary operation $\boxed{\circ}$ as the following (for the case of division, we assume $y \neq 0$):

$$x \boxed{\circ} y := \text{fl}(x \circ y),$$

then for any $x, y \in \mathbb{F}$, if $x \circ y \in \overline{\mathbb{F}}$, then $x \boxed{\circ} y = (x \circ y)(1 + \delta)$ with $|\delta| \le \text{u}$ from the Corollary 1.3.2.

Remark 1.4.1 (Cancellation error). *If $x, y \in \mathbb{R}$, then the relative error from the following binary operation $\text{fl}(x) \boxed{+} \text{fl}(y)$ can be estimated by*

$$\begin{aligned} \frac{|\text{fl}(x) \boxed{+} \text{fl}(y) - (x + y)|}{|x + y|} &\le \frac{|\text{fl}(x) \boxed{+} \text{fl}(y) - (\text{fl}(x) + \text{fl}(y))|}{|x + y|} + \frac{|(\text{fl}(x) + \text{fl}(y)) - (x + y)|}{|x + y|} \\ &\le \text{u} + (\text{u} + \text{u}^2)\frac{|x| + |y|}{|x + y|}. \end{aligned}$$

When x and y are close in magnitude but with opposite signs, the cancellation error will be significant.

1.4.1 Error Accumulation: Multiplication

For complicated computations on modern computers, the errors from arithmetic operations will accumulate towards the final result (we do not consider techniques such as fused multiply-add (FMA) here). To quantify the accumulation effect, we will need the following lemma.

Lemma 1.4.2. *For real numbers $a_1, a_2, \dots, a_n$ with $|a_k| \le \delta$ for $k = 1, \dots, n$, then for $n\delta < 1$, the following holds*

$$\prod_{k=1}^{n} (1 + a_k) = 1 + b_n,$$

where $|b_n| \le \frac{n\delta}{1-n\delta}$.

Proof. The proof is quite easy with induction. When $n = 1$, $|b_1| = |a_1| \le \delta \le \frac{\delta}{1-\delta}$. Suppose the claim holds for $n = m$, then for $n = m + 1$, we could see that

$$\prod_{k=1}^{m+1} (1 + a_k) = (1 + b_m)(1 + a_{m+1}) = 1 + b_{m+1},$$

which implies that $b_{m+1} = b_m + a_{m+1} + a_{m+1}b_m$, with the given bounds on a_{m+1} and b_m, we can estimate

$$|b_{m+1}| = |b_m + a_{m+1} + a_{m+1}b_m| \le \frac{(m+1)\delta}{1 - m\delta} \le \frac{(m+1)\delta}{1 - (m+1)\delta}.$$

□

Next, we consider the naive floating point product P_n of n real numbers $\{x_j\}_{j=1}^n \subset \mathbb{R}$ with assumption that $(2n-1)\mathrm{u} < 1$ by the following iteration

$$\begin{cases} P_k = \mathrm{fl}(x_1) & k = 1, \\ P_k = P_{k-1} \boxed{*}\, \mathrm{fl}(x_k) & k \ge 2. \end{cases}$$

Let $\mathrm{fl}(x_k) = x_k(1 + \tau_k)$, then $|\tau_k| \le \mathrm{u}$. From the n-th iteration step

$$P_n = P_{n-1} \boxed{*}\, \mathrm{fl}(x_k) = \mathrm{fl}(\mathrm{fl}(P_{n-1})\mathrm{fl}(x_n)) = \mathrm{fl}(P_{n-1})\mathrm{fl}(x_n)(1 + \delta_n)$$

such that $|\delta_n| \le \mathrm{u}$, since $P_{n-1} \in \mathbb{F}$, $\mathrm{fl}(P_{n-1}) = P_{n-1}$, then

$$\begin{aligned} P_n &= P_{n-1}\mathrm{fl}(x_n)(1 + \delta_n) = P_{n-2}\mathrm{fl}(x_{n-1})(1 + \delta_{n-1})(1 + \delta_k) = \cdots \\ &= \mathrm{fl}(x_1)\mathrm{fl}(x_2)\cdots\mathrm{fl}(x_n)\prod_{j=2}^{n}(1 + \delta_j) = \prod_{j=1}^{n} x_j(1 + \tau_j)\prod_{j=2}^{n}(1 + \delta_j) \\ &\le (1 + \eta_n)\prod_{j=1}^{n} x_j, \end{aligned}$$

where $|\eta_n| \le \frac{(2n-1)\mathrm{u}}{1-(2n-1)\mathrm{u}}$ by Lemma 1.4.2.

1.4.2 Error Accumulation: Addition

For naive floating point summation S_n of n real numbers $\{x_j\}_{j=1}^n$ by the iteration

$$\begin{cases} S_k = 0 & k = 0, \\ S_k = S_{k-1} \boxed{+} \text{fl}(x_k) & k \geq 1, \end{cases}$$

we can carry out a similar analysis. Let $S_j^* = \sum_{k=1}^j x_k$ and $\text{fl}(x_k) = x_k(1+\tau_k)$ for $|\tau_k| \leq \mathrm{u}$, denote $\Delta S_j = S_j^* - S_j$, then

$$\begin{aligned} \Delta S_j = S_j^* - S_j &= S_j^* - (S_{j-1} \boxed{+} \text{fl}(x_j)) \\ &= \Delta S_{j-1}(1+\delta_j) - \delta_j S_j^* - x_j \tau_j (1+\delta_j), \end{aligned}$$

where $|\tau_j|, |\delta_j| \leq \mathrm{u}$. Therefore

$$\begin{aligned} |\Delta S_n| &\leq |\Delta S_{n-1}|(1+\mathrm{u}) + \mathrm{u} \sum_{k=1}^{n} |x_k| + |x_n| \mathrm{u}(1+\mathrm{u}) \\ &\leq |\Delta S_{n-2}|(1+\mathrm{u})^2 + \mathrm{u}(1+\mathrm{u}) \sum_{k=1}^{j} |x_k| + |x_j| \mathrm{u}(1+\mathrm{u})^2 \\ &\quad + \mathrm{u} \sum_{k=1}^{j-1} |x_k| + |x_{j-1}| \mathrm{u}(1+\mathrm{u}) \\ &\leq \cdots \\ &\leq \sum_{l=1}^{n} \left(\mathrm{u}(1+\mathrm{u})^{l-1} \sum_{k=1}^{l} |x_k| + |x_l| \mathrm{u}(1+\mathrm{u})^{l-1} \right) \\ &\leq \sum_{l=1}^{n} \left(\mathrm{u}(1+\mathrm{u})^{l-1} \sum_{k=1}^{n} |x_k| \right) + \left(\sum_{l=1}^{n} |x_l| \right) \mathrm{u}(1+\mathrm{u})^{n-1} \\ &= \left(\sum_{l=1}^{n} |x_l| \right) ((1+\mathrm{u})^n - 1). \end{aligned}$$

Using the previous Lemma 1.4.2 will provide an estimate of $((1+\mathrm{u})^n - 1)$ as long as $n\mathrm{u} < 1$.

1.5 Exercises

Assume $n \in \mathbb{N}$ and $n\mathrm{u} < 1$, let $\{x_j\}_{j=1}^n$ be a sequence of real numbers, $\text{fl} : \overline{\mathbb{F}} \mapsto \mathbb{F}$ is the rounding operation, u is the machine epsilon. In the following, we briefly discuss the rounding error for floating point summation (sum reduction).

Definition 1.5.1. *A reduction* Π *of the floating point summation*

$$\text{fl}(x_1) \boxed{+} \text{fl}(x_2) \boxed{+} \cdots \boxed{+} \text{fl}(x_n)$$

is an evaluation order for the $\boxed{+}$ operations. For example, $n = 4$, then the reduction $\Pi = (3, 1, 2)$ is corresponding to the following calculation

$$\mathrm{fl}(x_1) \boxed{+} \mathrm{fl}(x_2) + \underbrace{(\mathrm{fl}(x_3) \boxed{+} \mathrm{fl}(x_4))}_{=y_1} \to \underbrace{(\mathrm{fl}(x_1) \boxed{+} \mathrm{fl}(x_2))}_{=y_2} \boxed{+} y_1 \to \underbrace{(y_2 \boxed{+} y_1)}_{=y_3},$$

where y_i denotes the result of i-th $\boxed{+}$ operation in the reduction. Obviously the final summation will be y_{n-1}. We denote $T_n(\Pi)$ be the result of floating point summation with reduction order Π.

1.5.1 Theoretical Part

Problem 1.5.2. *Prove that the native summation has following error bound*

$$|T_n(\Pi) - S_n| \le \left(\frac{\mathrm{u}n}{1 - \mathrm{u}n}\right) \sum_{j=1}^{n} |x_j|, \tag{1.1}$$

where $S_n = \sum_{j=1}^{n} x_j$ and $\Pi = (1, 2, \dots, (n-1))$.

Problem 1.5.3. *Prove that*

$$\min_{\Pi} |T_n(\Pi) - S_n| \le \left(\frac{\mathrm{u}H}{1 - \mathrm{u}H}\right) \sum_{j=1}^{n} |x_j|, \tag{1.2}$$

where $H = \lceil \log_2 n \rceil$ and $T_n(\Pi)$ is the floating point summation with reduction order Π.

Problem 1.5.4 (Horner's scheme). *The evaluation of polynomial*

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n$$

is mostly using Horner's scheme, which writes the polynomial in 'nested' form:

$$p(x) = a_0 + x(a_1 + x(a_2 + \dots x(a_{n-1} + x(a_n)))). \tag{1.3}$$

Please find an upper bound of the rounding error for this scheme.

1.5.2 Computational Part

Problem 1.5.5 (Pairwise summation). *Based on theoretical part, implement an algorithm for the summation $\sum_{j=1}^{n} x_j$ which has $\mathcal{O}(\mathrm{u} \log_2 n)$ rounding error.*

Problem 1.5.6 (Kahan compensated summation). *Suppose $a, b \in \mathbb{R}$, the rounding error for the sum $s = \mathrm{fl}(\mathrm{fl}(a) + \mathrm{fl}(b))$ ($a \ge b$) can be computed using*

$$\mathrm{fl}(\mathrm{fl}(s - \mathrm{fl}(a)) - \mathrm{fl}(b))$$

Based on this property, one can keep tracking of the rounding error. Implement the Algorithm 1 described below and compare the accuracy with naive summation and pairwise summation with test cases.

Problem 1.5.7. *Suppose the inputs $\{x_j\}_{j=1}^{n} \subset \mathbb{R}$ are randomly distributed (say $x_j \sim U(0, 1)$ i.i.d), what is growth of the expected rounding error with respect to total number of inputs n for the naive summation and pairwise summation? Please provide an explanation of your result. You can use Algorithm 1 as the accurate result approximately.*

Algorithm 1: Kahan compensated summation

```
Data: $\{x_j\}_{j=1}^{n} \subset \mathbb{R}$
Result: $s_n = \sum_{j=1}^{n} x_j$
$j \leftarrow 1$, $e_j \leftarrow 0$, $s_j \leftarrow x_j$ // initialization;
while $j < n$ do
    $j \leftarrow j + 1$
    $y_j = x_j - e_{j-1}$ //remove compensated error;
    $s_j = s_{j-1} + y_j$ //perform summation;
    $e_j = (s_j - s_{j-1}) - y_j$ //restore the rounding error;
end
```

Chapter 2

Interpolation

The interpolation solves problems of the following type:

> Given a set of predefined functions $\mathcal{K}$, find an element $f : \mathbb{I} \mapsto \mathbb{R}$ in $\mathcal{K}$ such that $y_j = f(x_j)$ for all $j = 0, \dots, n$.

Here $\mathbb{I}$ denotes a finite or infinite interval such that $x_1, \dots x_n \in \mathbb{I}$. One of the important applications for interpolation is Computer-Aided Design (CAD) which is used extensively in the manufacturing industry. Generally speaking, the interpolation provides a closed form of the function to determine the value of y where the parameter x is not accessible.

2.1 Polynomial Interpolation

The polynomial interpolation considers the set $\mathcal{K} = \Pi_m$, where the set Π_m represents the polynomials of with degree $\le m$. We will seek for a polynomial $f(x)$ with the constraints that

$$\begin{cases} f \in \mathcal{K} = \Pi_n, \\ f(x_k) = y_k \qquad \text{for } k = 0, 1, \dots, n. \end{cases}$$

The points x_k are called *interpolation nodes*, if $m > n$ (resp. $m < n$), the problem is underdetermined (resp. overdetermined). For the case that $m = n$, we have

Theorem 2.1.1. *There exists a unique polynomial function $f \in \Pi_n$ such that $f(x_j) = y_j$ for $j = 0, \dots, n$.*

Proof. Existence: In order to construct the polynomial f, it is straightforward to consider the general form of polynomial $f(x) = \sum_{j=0}^{n} a_j x^j$, then we can formulate a linear system for the coefficients a_j, $j = 0, \dots, n$, which is

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The matrix

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

is called *Vandermonde matrix.* To determine the coefficients a_j, one needs the matrix V to be invertible. Its determinant can be computed (try to prove it) as

$$\det(V) = \prod_{0 \le i \le j \le n} (x_j - x_i). \tag{2.1}$$

When x_j are distinct, the determinant is nonzero.
Uniqueness: Suppose there are two distinct polynomials $f, g \in \Pi_n$ satisfying the condition that $f(x_j) = g(x_j) = y_j$, then $f - g$ has $(n+1)$ roots x_j, $j = 0, \dots, n$. If $f \neq g$, it is clear that $f - g \in \Pi_n$ has at most n roots. Contradiction. □

In the above proof, the interpolation polynomial can be uniquely determined by solving the linear system

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

However, it is generally easier to compute the polynomial f with the *Lagrange polynomial interpolation* (which is somewhat equivalent to computing the inverse of V).

2.1.1 Lagrange Polynomial

Definition 2.1.2. *For the given distinct x_j, $j = 0, 1, \dots, n$, the $(n+1)$ Lagrange polynomials $L_0, L_1, \dots, L_n \in \Pi_n$ are defined by*

$$L_j(x) = \prod_{s=0, s \neq j}^{n} \frac{x - x_s}{x_j - x_s}, \quad j = 0, 1, \dots, n. \tag{2.2}$$

It is clear that these polynomials satisfy the conditions that

$$L_j(x_k) = \delta_{jk} := \begin{cases} 1 & \text{for } k = j, \\ 0 & \text{for } k \neq j. \end{cases} \tag{2.3}$$

Therefore these polynomials are linearly independent, which form a basis of the $(n+1)$-dimensional space Π_n.

Theorem 2.1.3. *The unique interpolating polynomial f satisfying $f(x_j) = y_j$, $j = 0, 1, \dots, n$ can be represented by*

$$f(x) = \sum_{j=0}^{n} y_j L_j(x). \tag{2.4}$$

Proof. It is straightforward to check the interpolation conditions are satisfied. □

Remark 2.1.4. *We introduce a preliminary procedure to compute the value of the interpolating polynomial f at a point x. Let constants k_j and $q(x)$ be defined as*

$$k_j = \prod_{s=0, s\neq j} \frac{1}{x_j - x_s}, \quad q(x) = \prod_{j=0}^{n} (x - x_j), \tag{2.5}$$

then

$$f(x) = \sum_{j=0}^{n} y_j L_j(x) = q(x) \sum_{j=0}^{n} k_j y_j \frac{1}{x - x_j}. \tag{2.6}$$

One can first compute k_j with $\mathcal{O}(n^2)$ flops, then $f(x)$ can be computed with $\mathcal{O}(n)$ flops. The advantage of the above scheme is the constants k_j are independent of y_j, therefore evaluating another instance of the interpolating polynomial will not need to re-compute them. The disadvantage is that if we add a new node, the constants k_j have to be updated with an additional cost of $\mathcal{O}(n)$ flops. Later we will see the Newton's form can overcome this issue.

2.1.2 Interpolation Error

When the data pairs (x_j, y_j), $j = 0, 1, \dots, n$ are generated by a sufficiently smooth function $h(x)$, it is possible to quantify the error between the interpolating polynomial $f(x)$ and $h(x)$.

Theorem 2.1.5. *Let $h : [a, b] \mapsto \mathbb{R}$ be a $(n + 1)$-times differentiable function. If $f(x) \in \Pi_n$ is the interpolating polynomial that*

$$f(x_j) = h(x_j),$$

for $j = 0, 1, \dots, n$. Then for each $\overline{x} \in [a, b]$, the error can be represented by

$$h(\overline{x}) - f(\overline{x}) = \frac{\omega(\overline{x})}{(n+1)!} h^{(n+1)}(\xi), \tag{2.7}$$

where $\xi = \xi(\overline{x}) \in [a, b]$ and $\omega(x) = \prod_{j=0}^{n}(x - x_j)$.

Proof. The proof is based on the Rolle's Theorem. Select any $\overline{x} \in [a, b]$ such that $\omega(\overline{x}) \neq 0$, then let

$$\psi(x) = h(x) - f(x) - k\omega(x),$$

the constant k is chosen such that $\psi(\overline{x}) = 0$. Then $\psi(x) = 0$ at $(n + 2)$ points

$$x_0, x_1, \dots, x_n, \overline{x} \in [a, b].$$

By Rolle's Theorem, $\psi^{(n+1)}$ has at least one zero ξ in $[a, b]$. Therefore

$$\psi^{(n+1)}(\xi) = h^{(n+1)}(\xi) - 0 - k(n + 1)! = 0. \tag{2.8}$$

□

Corollary 2.1.6. *If $h(x) \in C^{\infty}([a,b])$ satisfies that $\max_{x\in[a,b]} |h^{(n)}(x)| \le M < \infty$ for all $n \ge 0$, then the interpolating polynomial approximates h uniformly as the number of nodes $n \to \infty$.*

Proof. Since $|x - x_j| \le b - a$, the error is bounded by $\frac{(b-a)^{n+1}}{(n+1)!}M$, which converges to zero. □

It is interesting to think about the converse: under what kind of condition the interpolation error is not vanishing as the number of nodes tends to infinity? From the Theorem 2.1.5, the error depends on the sizes of three terms.

1. The bound of the $(n+1)$-th derivative, $\max_{x\in[a,b]} |h^{(n+1)}(x)|$. This could grow rapidly. For instance, $h(x) = x^{-1/2}$ on $[1/2, 3/2]$,
$$h^{(n+1)}(x) = \frac{(-1)^{n+1}}{2^{n+1}}(2n+1)!!x^{-(2n+3)/2}. \tag{2.9}$$
2. The function $\omega(x) = \prod_{j=0}^{n}(x - x_j)$, such product could be large if x and the nodes x_j are not so close.
3. The term $\frac{1}{(n+1)!}$, which decays fast.

We can see that for the function $h(x) = x^{-1/2}$ on $[1/2, 3/2]$, it is not trivial to show the interpolating polynomial could converge to h anymore (although it is still true for certain choices of x_j, see Exercise 2.5.2).

Next, we try to provide a better estimate of ω for the special choice: equally spaced nodes. Let the nodes $x_j = a + j\Delta$, where $\Delta = \frac{b-a}{n}$. It is not difficult (prove it) to see $\omega(x)$ will be the worst if x is located on the end sub-intervals, $[x_0, x_1]$ and $[x_{n-1}, x_n]$. Without loss of generality, we assume x is located on $[x_0, x_1]$, then
$$|x - x_j| \le (j+1)\Delta$$
for $j = 0, 1, \ldots, n$, which implies
$$|\omega(x)| \le \prod_{j=0}^{n} |x - x_j| \le (n+1)!\Delta^{n+1} = (n+1)!\frac{(b-a)^{n+1}}{n^{n+1}}, \tag{2.10}$$
use the Stirling approximation, $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, thus
$$(n+1)!\frac{(b-a)^{n+1}}{n^{n+1}} \sim \sqrt{2\pi n}\left(\frac{b-a}{e}\right)^{n+1}.$$
One should note that if the interval $[a,b]$ is shorter than e, then the above estimate is exponentially small (times $\sqrt{n}$), while it grows exponentially if $|b-a| > e$. Such an estimate is useful to derive uniform convergence.

Example 2.1.7. *Consider $h(x) = x^{-1}$ on $[1/2, 3/2]$. Then $h^{(n+1)}(x) = \frac{(n+1)!(-1)^{n+1}}{x^{n+2}}$, hence*
$$\frac{|h^{(n+1)}(x)|}{(n+1)!} \le \max_{x\in[1/2,3/2]} \left|\frac{1}{x^{n+2}}\right| = 2^{n+2} \tag{2.11}$$
and the upper bound of $|\omega|$ is $\mathcal{O}\left(\sqrt{n}\frac{1}{e^{n+1}}\right)$, therefore the interpolation error is $\mathcal{O}\left(\sqrt{n}\left(\frac{2}{e}\right)^{n+1}\right)$ converges to zero exponentially. It is important to notice that the above method only works for short intervals, if $b - a > \frac{e}{2}$, then we require more sophisticated estimates.

2.1.3 Runge's Phenomenon

From the above discussion, we can see there is a possibility that $\max_{x\in[a,b]} |h^{n+1}(x)|\omega(x)$ grows faster than $(n+1)!$, which would lead to divergence. Hence increasing the number of interpolation nodes (at least for equally spaced nodes) is not guaranteed for better approximation. The most famous example is the one made by Carl Runge.

$$h(x) = \frac{1}{1+x^2}, \quad x \in [-5,5]. \tag{2.12}$$

It can be shown that the interpolation will diverge at around 3.6 as $n \to \infty$ and the maximum

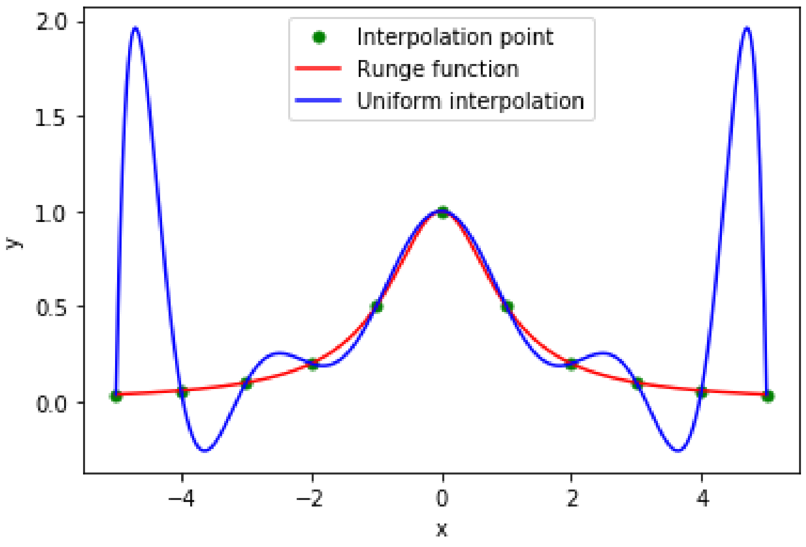

Figure 2.1: Runge's phenomenon. Interpolation with 11 equally spaced nodes.

error $\max_{x\in[-5,5]} |f_n(x) - h(x)|$ grows exponentially, where f_n is the interpolating polynomial with $n+1$ equally spaced nodes. There exist better choices of interpolation nodes to prevent such a phenomenon. We will discuss this topic in Section 2.1.4.

2.1.4 Chebyshev Interpolation

The Chebyshev interpolation aims to minimize the bound of the interpolation error. The bound of $\omega(x)$ only depends on the choice of the nodes, so a natural question is: what kind of interpolation nodes will *minimize* $\max_{x\in[a,b]} \prod_{j=0}^{n} |x - x_j|$. We first restrict our analysis to the interval $[a,b] = [-1,1]$ for simplicity, the general case will be discussed later.

Example 2.1.8. *When* $n = 1$, $\omega(x) = (x - x_0)(x - x_1)$, *this function changes sign over the subintervals* $[-1, x_0)$, (x_0, x_1), $(x_1, 1]$, *thenthe we can compute the maximum of* $|\omega(x)|$ *on these subintervals. Therefore we need to solve*

$$\min_{x_0,x_1\in[-1,1]} \max\left((1+x_0)(1+x_1), \frac{(x_1-x_0)^2}{4}, (1-x_0)(1-x_1)\right), \tag{2.13}$$

while we can observe that

$$\frac{1}{2}(1+x_0)(1+x_1) + \frac{(x_1-x_0)^2}{4} + \frac{1}{2}(1-x_0)(1-x_1) = 1 + \frac{(x_0+x_1)^2}{4} \geq 1 \tag{2.14}$$

holds for any choice of x_0, x_1, which means the maximum is at least $1/2$, it occurs when all terms are equal and $x_0 + x_1 = 0$. Hence $x_0 = -\sqrt{2}/2, x_1 = \sqrt{2}/2$.

Definition 2.1.9. *The Chebyshev polynomials of the first kind are defined by:*

$$T_k(x) = \cos(k \arccos x), \quad x \in [-1,1].$$

Theorem 2.1.10. *The Chebyshev polynomial satisfies the following:*

1. $T_k(\cos\theta) = \cos k\theta, \quad \theta \in [0,\pi]$.
2. $T_0 \equiv 1, T_1(x) = x$ *and*
$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad k \geq 1.$$
3. $\max_{x\in[-1,1]} |T_k(x)| = 1$.
4. *The leading coefficient of $T_k(x)$ is 2^{k-1}.*
5. *T_k has a total of $(k+1)$ extrema $s_j = \cos(\frac{j\pi}{k}), j = 0, 1, \dots, n$ in the interval $[-1,1]$ such that $T_k(s_j) = (-1)^j$.*

Proof. The first three statements are straightforward after replacing the variable $x = \cos\theta$. The fourth statement is an immediate result with induction through the recursion formula $T_{k+1}(x) = 2xT_k(x) - T_{k-1}$. The last statement is trivial. □

More importantly, the Chebyshev polynomial has the following optimality property.

Theorem 2.1.11. *The optimal choice of interpolation nodes $\{x_j\}_{j=0}^n$ that minimize $\max|\omega(x)|$ is the extrema of Chebyshev polynomial T_{n+1}.*

$$\min_{x_j\in[-1,1]} \max_{x\in[-1,1]} |\omega(x)| = \max_{x\in[-1,1]} \frac{1}{2^n}|T_{n+1}(x)| = \frac{1}{2^n} \tag{2.15}$$

Proof. Let the roots of $T_{n+1}(x)$ be $z_0, z_1, \dots, z_n \in [-1,1]$, then we can write

$$T_{n+1} = 2^n(x-z_0)(x-z_1)\dots(x-z_n),$$

therefore $\frac{1}{2^n}T_{n+1}(x)$ is a polynomial with leading coefficient as 1. Since $\max_{x\in[-1,1]}|T_{n+1}(x)| = 1$, it is clear that $\max_{x\in[-1,1]} \frac{1}{2^n}|T_{n+1}(x)| = \frac{1}{2^n}$, which is the second equality in (2.15). For the first equality, we try to prove it by contradiction. Let $x_0, x_1, \dots, x_n \in [-1,1]$, such that

$$\max_{x\in[-1,1]} |\omega(x)| < \frac{1}{2^n},$$

then we define the polynomial $\psi(x) = \frac{1}{2^n}T_{n+1}(x) - \omega(x)$, its degree is at most n due to cancellation, therefore at most have n zeros. On the other hand, because $\frac{1}{2^n}T_{n+1}(s_j) = \frac{1}{2^n}(-1)^j$ at the extrema $s_j = \cos(\frac{j\pi}{n+1})$, $j = 0, \dots, (n+1)$, the polynomial $\psi(s_j)$ must share the same sign of $\frac{1}{2^n}T_{n+1}(s_j)$. This means $\psi(x)$ changes sign $(n+1)$ times, hence $(n+1)$ zeros. It is a contradiction. □

Definition 2.1.12. *The interpolation nodes $z_j = \cos(\frac{(2j+1)\pi}{2(n+1)})$, $j = 0, 1, \ldots, n$ are called "Chebyshev nodes". These nodes are the zeros of Chebyshev polynomial T_{n+1}.*

Now we can generalize the above theorem to any interval $[a, b]$. One can defined the affine transformation ϕ mapping $[-1, 1]$ to $[a, b]$ by $\phi(x) = \frac{1}{2}(a + b + (b - a)x)$. It is not difficult to prove the following.

Corollary 2.1.13. *The optimal choice of interpolation nodes that minimize* $\max |\omega(x)|$ *on* $[a, b]$ *are* $\phi(z_j)$ *and*

$$\min_{x_j \in [a,b]} \max_{x \in [a,b]} |\omega(x)| = \frac{(b-a)^{n+1}}{2 \cdot 4^n}.$$

This bound is much smaller than the bound for equally spaced nodes.

2.1.5 Stability of Polynomial Interpolation

Suppose there is some perturbation of the data $\tilde{y}_j = y_j + \varepsilon_j$ at the interpolation node x_j. Let $\tilde{f}_n(x)$ and $f_n(x)$ be the interpolating polynomials on perturbed data and original data. Then with Lagrange polynomials,

$$\begin{aligned} |f_n(x) - \tilde{f}_n(x)| &= |\sum_{j=0}^{n} (y_j - \tilde{y}_j) L_j(x)| \\ &\leq \left(\max_j |\varepsilon_j|\right) \sum_{j=0}^{n} |L_j(x)|. \end{aligned} \tag{2.16}$$

Here $\lambda_n(x) := \sum_{j=0}^{n} |L_j(x)|$ is the *Lebesgue function.* It is a piecewise-defined polynomial. Its maximum Λ_n is the *Lebesgue constant* and it only depends on the choice of interpolation nodes. For the equally spaced nodes, this Lebesgue constant grows exponentially,

$$\Lambda_{n,\text{equal}} \sim \frac{2^{n+1}}{en \log n}. \tag{2.17}$$

For the general case, it has been proved by Paul Erdös (1964) that there exists a constant $C > 0$

$$\Lambda_n > \frac{2}{\pi} \log(n+1) - C, \quad n \geq 0. \tag{2.18}$$

As the number of nodes $n \to \infty$, $\Lambda_n \to \infty$. This leads to the result of Faber that for any choice of nodes, there exists a continuous function not able to be approximated by the interpolating polynomial. The Chebyshev nodes are almost optimal in the sense that

$$\Lambda_{n,\text{Chebyshev}} < \frac{2}{\pi} \log(n+1) + 1. \tag{2.19}$$

The set of nodes that minimizes Λ_n is difficult to compute. A slightly better set of nodes than Chebyshev nodes is the *extended Chebyshev nodes*:

$$\tilde{x}_j = \frac{\cos\left(\frac{2j+1}{2(n+1)\pi}\right)}{\cos\left(\frac{\pi}{2(n+1)}\right)}. \tag{2.20}$$

2.1.6 Newton Form

The Newton form is useful when we dynamically add interpolation nodes. Consider the following scenario: we already have found an interpolation polynomial f_k through the data (x_0, y_0), (x_1, y_1),..., (x_k, y_k), then if an addition pair (x_{k+1}, y_{k+1}) is provided, how to effectively transform f_k to f_{k+1}? If we write

$$f_{k+1}(x) = f_k(x) + c_{k+1}(x - x_0)(x - x_1)\dots(x - x_k),$$

then $f_{k+1}(x_j) = f_k(x_j)$, $j = 0, 1, \dots, k$. Therefore we only need to take care of the equality $f_{k+1}(x_{k+1}) = y_{k+1}$, which means

$$c_{k+1} = \frac{y_{k+1} - f_k(x_{k+1})}{\prod_{j=0}^{k}(x_{k+1} - x_j)}. \tag{2.21}$$

Such an inductive procedure produces the Newton form:

$$f_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \dots + c_n(x - x_0)\dots(x - x_{n-1}). \tag{2.22}$$

where the constant c_j depends on $x_0, x_1, \dots, x_j$ only. The polynomials $\prod_{j=0}^{k}(x - x_j)$ are called *Newton polynomials*. When the coefficients c_k are known, the Newton form (2.22) can be evaluated by the famous "Horner's scheme" (see Exercise 1.5.4), which is

$$f_n(x) = c_0 + (x - x_0)(c_1 + (x - x_1)(c_2 + (x - x_2)(c_3 + \dots))), \tag{2.23}$$

the evaluation order starts from the innermost part $c_n(x - x_{n-1})$. This formulation has a complexity of $3n$ flops.

Remark 2.1.14. *The computation of c_k is not cheap from (2.21). A naive algorithm with Horner's scheme roughly takes $5n^2/2 + \mathcal{O}(n)$ flops to compute all coefficients. The "divided differences" is a better way to compute c_k.*

Definition 2.1.15. *Let the interpolation nodes be $\{x_0, x_1, \dots, x_n\}$, the "divided differences" are defined recursively as follows (the square bracket is used to distinguish from the usual bracket):*

$$\begin{aligned} f[x_j] &:= f(x_j), \\ f[x_j, \dots, x_{j+k}] &:= \frac{f[x_{j+1}, \dots, x_{j+k}] - f[x_j, \dots, x_{j+k-1}]}{x_{j+k} - x_j}, \end{aligned} \tag{2.24}$$

where $0 \le j, k \le n$ and $j + k \le n$.

The following example graph helps understand the relationships among the divided differences.

$$\begin{tikzcd}
f[x_0] \arrow[dr] & & & & \\
f[x_1] \arrow[r] \arrow[dr] & f[x_0,x_1] \arrow[dr] & & & \\
f[x_2] \arrow[r] \arrow[dr] & f[x_1,x_2] \arrow[r] \arrow[dr] & f[x_0,x_1,x_2] \arrow[dr] & & \\
f[x_3] \arrow[r] \arrow[dr] & f[x_2,x_3] \arrow[r] \arrow[dr] & f[x_1,x_2,x_3] \arrow[r] \arrow[dr] & f[x_0,x_1,x_2,x_3] \arrow[dr] & \\
f[x_4] \arrow[r] & f[x_3,x_4] \arrow[r] & f[x_2,x_3,x_4] \arrow[r] & f[x_1,x_2,x_3,x_4] \arrow[r] & f[x_0,x_1,x_2,x_3,x_4]
\end{tikzcd}$$

(2.25)

Computing all of the divided differences requires $3n^2/2+\mathcal{O}(n)$ flops. The following theorem is the main statement for the Newton form.

Theorem 2.1.16. *The interpolation polynomial f_n in Newton form is given by*

$$f_n(x) = f[x_0] + f[x_0,x_1](x-x_0) + \cdots + f[x_0,\dots,x_n](x-x_0)(x-x_1)\dots(x-x_{n-1}). \tag{2.26}$$

In other words, $c_k = f[x_0,\dots,x_k]$.

Proof. We prove this by induction. Assume the statement is true for n and interpolation node and corresponding values $(x_0, f[x_0]), (x_1, f[x_1]), \dots, (x_n, f[x_n])$. For a new node and value $(x_{n+1}, f[x_{n+1}])$, it is known from (2.21) that c_{n+1} is the coefficient of leading power. Let g_n be the interpolation polynomial in Newton form through nodes $(x_1, f[x_1]), \dots, (x_{n+1}, f[x_{n+1}])$, then

$$\psi(x) := g_n(x)(x-x_0) - f_n(x)(x-x_{n+1})$$

satisfies that $\psi(x_j) = f[x_j](x_{n+1}-x_0)$ for $0 \le j \le n+1$. Therefore

$$f_{n+1}(x) = \frac{g_n(x)(x-x_0) - f_n(x)(x-x_{n+1})}{x_{n+1}-x_0}. \tag{2.27}$$

The leading power's coefficient is then

$$\frac{f[x_1,\dots,x_{n+1}] - f[x_0,\dots,x_n]}{x_{n+1}-x_0} = f[x_0,x_1,\dots,x_{n+1}]. \tag{2.28}$$

□

Remark 2.1.17. *The divided difference $f[x_j,\dots,x_{j+k}]$ is the coefficient of leading power of the interpolating polynomial through $(x_j, f[x_j]), \dots, (x_{j+k}, f[x_{j+k}])$. It can be shown that*

$$f[x_j,\dots,x_{j+k}] = \frac{1}{k!} f^{(k)}(\xi)$$

for some $\xi \in [a,b]$. See Exercise 2.5.4.

Remark 2.1.18. *The error estimate can be derived as*

$$f(x) - f_n(x) = f[x_0, x_1, \dots, x_n, x](x - x_0)\dots(x - x_n). \tag{2.29}$$

Remark 2.1.19. *The Newton form* (2.22) *does not require distinct nodes. The divided difference can be defined as a limit for repeated nodes:*

$$f[x_0, x_0] = \lim_{x_1 \to x_0} \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f'(x_0). \tag{2.30}$$

Moreover, using Taylor expansion, $f[\underbrace{x_0, \dots, x_0}_{(k+1)\ \textit{times}}] = \frac{1}{k!} f^{(k)}(x_0)$*. However, in such cases, the divided differences are not possible to be computed if the derivative values are not provided. We will discuss this scenario later in Hermite interpolation polynomial, see Section* 2.1.7*.*

Remark 2.1.20. *The algorithm to compute the divided difference can be made more efficient with a single column to store the diagonal elements.* $\rightsquigarrow$ *represents the number that is not changing.*

$$\begin{tikzcd}
f[x_0] \arrow[r, squiggly] \arrow[dr] & f[x_0] \arrow[r, squiggly] & f[x_0] \arrow[r, squiggly] & f[x_0] \arrow[r, squiggly] & f[x_0] \\
f[x_1] \arrow[r] \arrow[dr] & f[x_0, x_1] \arrow[r, squiggly] \arrow[dr] & f[x_0, x_1] \arrow[r, squiggly] & f[x_0, x_1] \arrow[r, squiggly] & f[x_0, x_1] \\
f[x_2] \arrow[r] \arrow[dr] & f[x_1, x_2] \arrow[r] \arrow[dr] & f[x_0, x_1, x_2] \arrow[r, squiggly] \arrow[dr] & f[x_0, x_1, x_2] \arrow[r, squiggly] & f[x_0, x_1, x_2] \\
f[x_3] \arrow[r] \arrow[dr] & f[x_2, x_3] \arrow[r] \arrow[dr] & f[x_1, x_2, x_3] \arrow[r] \arrow[dr] & f[x_0, x_1, x_2, x_3] \arrow[r, squiggly] \arrow[dr] & f[x_0, x_1, x_2, x_3] \\
f[x_4] \arrow[r] & f[x_3, x_4] \arrow[r] & f[x_2, x_3, x_4] \arrow[r] & f[x_1, x_2, x_3, x_4] \arrow[r] & f[x_0, x_1, x_2, x_3, x_4]
\end{tikzcd} \tag{2.31}$$

2.1.7 Hermite Polynomial Interpolation

The Lagrange polynomial interpolation only requires the values of the data function h at each node. It can be generalized when the derivative values of h are also available.

Let the tuple $(h(x_j), h^{(1)}(x_j), \dots, h^{(m_j)}(x_j))$ be the provided derivative values at the interpolation node x_j, $j = 0, \dots, n$ and $m_j \geq 0$. $N = \sum_{j=0}^{n}(m_j + 1)$ is the total number of constraints. It can be shown that there exists a unique polynomial $H_{N-1} \in \Pi_{N-1}$ satisfies

$$H_{N-1}^{(k)}(x_j) = y_j^k := h^{(k)}(x_j), \quad j = 0, \dots, n, \quad 0 \leq k \leq m_j.$$

This polynomial is called *Hermite interpolation polynomial*. The idea to construct the Hermite interpolation polynomial borrows from the Lagrange polynomials, which is to find basis L_{jk} such that

$$\frac{d^p}{dx^p} L_{jk}(x_l) = \begin{cases} 1, & l = j, k = p \\ 0, & \text{otherwise}. \end{cases} \tag{2.32}$$

Once these polynomials are obtained, the Hermite interpolation is straightforward:

$$H_{N-1}(x) = \sum_{j=0}^{n} \sum_{k=0}^{m_j} y_j^k L_{jk}(x).$$

Its uniqueness can be concluded from the linear independence of the basis L_{jk}. However, the construction method in (2.32) is not the simplest. It is known that the Newton form (2.22) works for repeated nodes as long as the diagram's diagonal (2.25) can be filled. Therefore, we can arrange the nodes

$$\underbrace{x_0, \dots, x_0}_{(m_0+1)\text{ times}}, \quad \underbrace{x_1, \dots, x_1}_{(m_1+1)\text{ times}}, \quad \dots, \quad \underbrace{x_n, \dots, x_n}_{(m_n+1)\text{ times}}$$

In this way, all of the necessary divided differences can be computed.

Remark 2.1.21. *The error estimate for Hermite polynomial interpolation will be the same as the Newton form, see Remark 2.1.18.*

2.2 Trigonometric Interpolation

Periodic functions occur in many applications, that is, $f(x+T) = f(x), x \in \mathbb{R}$ for some $T > 0$. For example, a closed planar curve can be parameterized as a periodic function naturally. The polynomial interpolation does not suit periodic functions, this is because polynomials will eventually go to infinity as $x \to \infty$. The most used interpolation for the periodic function is the *trigonometric polynomial interpolation*. In the following, we assume the period $T = 2\pi$ without loss of generality.

2.2.1 Fourier Series

Definition 2.2.1. *For $n \geq 0$, we defined F_n the space of trigonometric polynomials*

$$F_n := \{f(x) \mid f(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos kx + \sum_{k=1}^{n} b_k \sin kx,\ a_k, b_k \in \mathbb{R}\}. \tag{2.33}$$

The coefficients $a_0, \dots, a_n$, $b_1, \dots, b_n$ can be also chosen as complex numbers. $f \in F_n$ is said to be of degree n if $|a_n| + |b_n| > 0$.

The concept of "degree" here can be validated by the addition theorem of trigonometric functions. For instance, if $f_1 \in F_k$, $f_2 \in F_l$, then $f_1 f_2 \in F_{k+l}$. In the next, we discuss the uniqueness of the interpolation with the trigonometric polynomial.

Lemma 2.2.2. *A trigonometric polynomial $f \in F_n$ that has more than $2n$ zeros in $[0, 2\pi)$ must vanish identically.*

Proof. Rewrite the trigonometric function in the form of

$$f_n(x) = \sum_{k=-n}^{n} \gamma_k e^{ikx}. \tag{2.34}$$

where $\gamma_0 = \frac{1}{2}a_0$ and $\gamma_k = \frac{1}{2}(a_k - ib_k)$ and $\gamma_{-k} = \frac{1}{2}(a_k + ib_k)$, $k = 1, \dots, n$. Then substitute $z = e^{ix}$ and set

$$p(z) = \sum_{k=-n}^{n} \gamma_k z^{n+k}, \tag{2.35}$$

one can rewrite $f_n(x) = z^{-n}p(z)$. If $f_n(x)$ has more than $2n$ zeros, then $p(z)$ has more than $2n$ zeros, which is a contradiction since $p(z)$ is a polynomial of degree $2n$. □

Remark 2.2.3. *Since* $\sin nx \in F_n$ *has* $2n$ *zeros* $\frac{\pi j}{n}$, $j = 0, \dots, 2n-1$, *it means to uniquely determine a trigonometric polynomial in* F_n, *exactly* $2n+1$ *values are needed. This is also known as the Nyquist sampling theorem.*

A direct corollary is the linear independence of the functions 1, $\cos kx$ and $\sin kx$, $k = 1, \dots n$, these $(2n+1)$ functions form a natural basis for the trigonometric polynomial space F_n.

Corollary 2.2.4. *The functions* $1, \cos kx, \sin kx, k = 1, \dots, n$ *are linearly independent on* $C([0, 2\pi])$, *hence* F_n *is a* $(2n+1)$ *dimensional space.*

To determine the coefficients a_k, b_k from $(2n+1)$ data paris (x_j, y_j), $j = 0, \dots, 2n$. We simply follow the idea of Lagrange polynomials by creating the basis polynomial $l_k(x)$ such that

$$l_k(x_j) = \begin{cases} 1, & j = k, \\ 0, & \text{otherwise}. \end{cases} \tag{2.36}$$

Remark 2.2.5. *A natural idea is replace* $x - x_j$ *in the Lagrange basis by* $\sin(x - x_j)$ *and produce something like*

$$\prod_{j=0, j\neq k}^{2n} \frac{\sin(x - x_j)}{\sin(x_k - x_j)}$$

but $\sin(x - x_j)$ *has two roots on* $[0, 2\pi)$, *therefore we need to rescale it to* $[0, \pi)$.

Theorem 2.2.6. *Let the basis trigonometric polynomial*

$$l_k(x) = \prod_{j=0, j\neq k}^{2n} \frac{\sin(\frac{x - x_j}{2})}{\sin(\frac{x_k - x_j}{2})},$$

then the interpolation trigonometric polynomial is

$$f_n(x) = \sum_{k=0}^{2n} y_k l_k(x). \tag{2.37}$$

Proof. It remains to show $l_k \in F_n$. This can be seen by splitting l_k into n pairs, each pair takes the form of

$$\sin(\frac{x-x_0}{2})\sin(\frac{x-x_1}{2}) = \frac{1}{2}\cos\left(\frac{x_0-x_1}{2}\right) - \frac{1}{2}\cos\left(\frac{2x-x_0-x_1}{2}\right) \in F_1. \tag{2.38}$$

□

Computationally, we can reuse the previously known barycentric form but there exist better methods. For simplicity, we consider the equal space nodes in the following (non-uniform nodes could achieve the same complexity though).

$$x_j = \frac{2\pi j}{2n+1}, \quad j = 0, \dots, 2n. \tag{2.39}$$

We will try to locate the coefficients γ_k in the complex form (see (2.34)) from the interpolation conditions.

$$f_n(x_j) = y_j = \sum_{k=-n}^{n} \gamma_k e^{ikx_j}. \tag{2.40}$$

Use the property of the functions e^{ikx_j} that

$$\sum_{k=0}^{2n} e^{ikx_j} = \begin{cases} 2n+1, & k=0 \\ 0, & \text{otherwise}. \end{cases} \tag{2.41}$$

It is not difficult to derive

$$\sum_{j=0}^{2n} y_j e^{-imx_j} = \sum_{k=-n}^{n} \gamma_k \sum_{j=0}^{2n} e^{i(k-m)x_j} = \gamma_m(2n+1). \tag{2.42}$$

Therefore, we can compute

$$\gamma_m = \frac{1}{2n+1}\sum_{j=0}^{2n} y_j e^{-imx_j}. \tag{2.43}$$

When the coefficients γ_m are known, a Horner's scheme can be employed to evaluate the trigonometric polynomial in $\mathcal{O}(n)$ time complexity. However, naive computing of all of the coefficients γ_k will cost $\mathcal{O}(n^2)$ flops. The fast Fourier transform can reduce the time complexity to $\mathcal{O}(n\log n)$.

Remark 2.2.7. *For even number of equally spaced nodes $x_j = \frac{j\pi}{n}$, $0 \le j \le 2n-1$, the basis* $\sin(nx)$ *equals to zero constantly. There are $2n$ coefficients to be determined only. We can derive a similar formula as (2.43) by replacing $(2n+1)$ with $2n$.*

2.2.2 Fast Fourier Transform

The discrete Fourier transform DFT of a vector $\mathbf{a} = (a_0, \dots, a_{n-1})$ is to evaluate the following vector:

$$\mathtt{DFT}(\mathbf{a})_k := \frac{1}{n}\sum_{j=0}^{n-1} a_j e^{-2\pi ijk/n}, \quad k = 0, \dots, n-1.$$

This is the exact formula to compute the coefficients for the trigonometric interpolation polynomial. Such transform is most efficiently calculated through the fast Fourier transform (`fft`). The fast Fourier transform exploits the symmetry in $e^{2\pi ij/n}$ when n is the power of two using divide-and-conquer. Let $\omega = e^{-2\pi i/n}$ and c_k be

$$c_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j \omega^{jk}, \quad k = 0, \dots, n-1. \tag{2.44}$$

Let $m = n/2 \in \mathbb{N}$, then $\omega^n = 1$, $\omega^m = -1$. We can separate c_k into two parts with even j and odd j.

$$c_k = \frac{1}{2}A_k + \frac{1}{2}B_k\omega^k, \quad c_{k+m} = \frac{1}{2}A_k - \frac{1}{2}B_k\omega^k \tag{2.45}$$

where

$$A_k = \frac{1}{m}\sum_{j=0}^{m-1} y_{2j}(\omega^2)^{jk}, \quad B_k = \frac{1}{m}\sum_{j=0}^{m-1} y_{2j+1}(\omega^2)^{jk}, \tag{2.46}$$

both A_k and B_k are in the same form and similar to c_k, but with only half of the terms in summation. This implies a recursive algorithm. Suppose A_k and B_k, $0 \le k \le m-1$ can be computed with $f(m)$ operations each, then

$$f(n) = f(2m) = 2f(m) + 4m \tag{2.47}$$

The second term includes $2m$ multiplications and $2m$ additions in (2.21). Therefore

$$\begin{aligned} f(n) &= 2f(\frac{n}{2}) + 2n \\ &= 4f(\frac{n}{4}) + 2n + 2n \\ &= \dots \\ &= nf(1) + \underbrace{2n + \dots + 2n}_{\log_2 n \text{ times}} = 2n\log_2 n. \end{aligned} \tag{2.48}$$

since $f(1) = 0$, no computation is needed in this case. The `fft` is usually a standard routine in modern scientific computing software.

2.2.3 Interpolation Error of Trigonometric Polynomial

The L^2 error estimate will be discussed at a later point. In this part, we only focus on the L^∞ error estimate.

Theorem 2.2.8. *If $f \in C^2(\mathbb{R})$ is a 2π-period function, then the trigonometric interpolation polynomial with $2n+1$ equally spaced nodes converges uniformly as the number of nodes tends to infinity.*

Proof. Since f is continuously differentiable, its Fourier series converges to f uniformly. Let $f(x), g(x)$ be

$$f(x) = \sum_{s=-\infty}^{\infty} \gamma_s e^{isx}, \quad g(x) = \sum_{s=-n}^{n} \gamma_s e^{isx},$$

respectively and denote $h(x) = f(x) - g(x) = \sum_{|s|>n} \gamma_s e^{isx}$ the reminder. Use integration by parts twice,

$$\gamma_s = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-isx}dx = -\frac{1}{2\pi}\frac{1}{s^2}\int_0^{2\pi} f''(x)e^{-isx}dx \le \frac{1}{2\pi s^2}\|f''\|_\infty. \tag{2.49}$$

On the other hand, the interpolation polynomial

$$\begin{aligned}
f_n(x) &= \sum_{m=-n}^{n}\left(\frac{1}{2n+1}\sum_{j=0}^{2n} y_j e^{-imx_j}\right)e^{imx} \\
&= \sum_{m=-n}^{n}\left(\frac{1}{2n+1}\sum_{j=0}^{2n} (g(x_j)+h(x_j))e^{-imx_j}\right)e^{imx} \\
&= \sum_{m=-n}^{n}\left(\frac{1}{2n+1}\sum_{j=0}^{2n} (\sum_{s=-n}^{n}\gamma_s e^{isx_j}+h(x_j))e^{-imx_j}\right)e^{imx} \\
&= g(x) + \sum_{m=-n}^{n}\left(\frac{1}{2n+1}\sum_{j=0}^{2n} h(x_j)e^{-imx_j}\right)e^{imx} \\
&= g(x) + \frac{1}{2n+1}\sum_{j=0}^{2n} h(x_j)\frac{\sin((2n+1)(x-x_j)/2)}{\sin((x-x_j)/2)}
\end{aligned} \tag{2.50}$$

Therefore

$$|f - f_n| \le \|h\|_\infty + \|h\|_\infty \frac{1}{2n+1}\sum_{j=0}^{2n}\left|\frac{\sin((2n+1)(x-x_j)/2)}{\sin((x-x_j)/2)}\right|. \tag{2.51}$$

It is simple to derive $\|h\|_\infty \le \sum_{|s|>n}|\gamma_s| \le \frac{1}{n\pi}\|f''\|_\infty$. We only need to estimate

$$\sum_{j=0}^{2n}\left|\frac{\sin((2n+1)(x-x_j)/2)}{\sin((x-x_j)/2)}\right|$$

Separate the nodes into two groups: The first group with $|x - x_j| < \frac{2\pi}{2n+1}$, the absolute value is bounded by $(2n+1)$, there are at most 3 nodes lying in this region, thus the contribution is at most $\mathcal{O}(n)$. The second group is $\pi \ge |x - x_j| \ge \frac{2\pi}{2n+1}$, while the rest is symmetric, then one can estimate

$$\left|\frac{\sin((2n+1)(x-x_j)/2)}{\sin((x-x_j)/2)}\right| \le \frac{\pi}{|x-x_j|} \tag{2.52}$$

where the inequality $\sin x \geq \frac{2}{\pi}x$ for $0 \leq x \leq \frac{\pi}{2}$, then this part will be at most $\mathcal{O}(n\log n)$, the total contribution is bounded by $\mathcal{O}(n\log n)$. Then (2.51) can be bounded by

$$|f - f_n| \leq \|f''\|_\infty \mathcal{O}\left(\frac{\log n}{n}\right) \to 0, \quad \text{as } n \to \infty. \tag{2.53}$$

□

The above result can be extended to the case of Hölder continuous function, see the work of Dunham Jackson (1913).

2.3 Spline Interpolation

It has been seen that increasing the number of interpolation nodes will not always help to improve the approximation. The spline interpolation is to conquer this issue by using the piecewise low-degree polynomials.

Definition 2.3.1. *Let $x_0, \dots, x_n$ be the distinct nodes on $[a, b]$ such that $a = x_0 < \cdots < x_n = b$. The piecewise defined function $s_k(x)$ on the interval $[a, b]$ is a spline of degree k to the nodes if*

$$s_k|_{[x_j, x_{j+1}]} \in \Pi_k, \quad s_k \in C^{k-1}([a, b]). \tag{2.54}$$

The spline function s_k is $(k-1)$-times continuously differentiable and piecewise polynomial of degree k.

Then the space of splines s_k will be $(n+k)$ dimension: each interval has $(k+1)$ dimensions, each interface imposes k constraints, therefore $n(k+1) - (n-1)k = n+k$ dimensions. This shows that to uniquely determine a spline on the nodes, we will require $n+1$ interpolation values and $k-1$ additional constraints. Usual choices are

1. periodic splines. $s_k^{(m)}(a) = s_k^{(m)}(b)$ for $m = 0, 1, \dots, k-1$.

2. natural splines. $s_k^{(l+j)}(a) = s_k^{(l+j)}(b) = 0$, $j = 0, 1, \dots, l-2$ and $k = 2l-1$ with $l \geq 2$.

In the following, we discuss some useful examples of spline.

2.3.1 Linear Splines

The linear splines are a special case of splines. It uses piecewise linear polynomials on each sub-interval and does not impose any derivative continuity. Let y_j be the interpolation values at nodes x_j, respectively. The interpolation has an explicit form:

$$s_1(x) = y_{j-1} + \frac{x - x_{j-1}}{x_j - x_{j-1}} y_j \tag{2.55}$$

on the interval $[x_{j-1}, x_j]$. It can represented as a linear combination of the "hat" function basis $\theta_j(x)$, defined by

$$\theta_j(x) = \begin{cases} \frac{x-x_{j-1}}{x_j-x_{j-1}}, & x \in [x_{j-1}, x_j], \quad 1 \leq j \leq n \\ \frac{x-x_{j+1}}{x_j-x_{j+1}}, & x \in [x_j, x_{j+1}], \quad 0 \leq j \leq n-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.56}$$

then $s_1(x)$ can be written as

$$s_1(x) = \sum_{j=0}^{n} \theta_j(x) y_j.$$

The interpolation error can be derived directly from the previous theory for two interpolation nodes. Let $f \in C^2([a, b])$, then on $[x_{j-1}, x_j]$, the interpolation error is

$$\frac{1}{2!} f''(\xi)(x - x_{j-1})(x - x_j) \leq \frac{1}{8} \|f''\|_\infty |x_j - x_{j-1}|^2. \tag{2.57}$$

Therefore the interpolation error over $[a, b]$ is $\frac{1}{8}\|f''\|_\infty h^2$, where $h = \max |x_j - x_{j-1}|$.

2.3.2 Cubic Splines

The cubic splines are particularly important. Let $a = x_0 < x_1 < \cdots < x_n = b$, and the corresponding values are y_j, $j = 0, \ldots, n$. The constraints for cubic splines are: piecewise polynomial of degree 3 and continuous second derivative. Denote the interpolation spline as s_3, then s_3'' is a piecewise linear function. On the sub-interval $[x_{j-1}, x_j]$, it can be represented by

$$s_3''(x) = M_{j-1} \frac{x_j - x}{h_j} + M_j \frac{x - x_{j-1}}{h_j}, \quad j = 1, \ldots, n, \tag{2.58}$$

where $h_j = x_j - x_{j-1}$, $M_j = s_3''(x_j)$. Integrating the above formula twice,

$$s_3(x) = M_{j-1} \frac{(x_j - x)^3}{6h_j} + M_j \frac{(x - x_{j-1})^3}{6h_j} + A_j(x - x_{j-1}) + B_j \tag{2.59}$$

The additional constants A_j, B_j can be determined by imposing $f(x_{j-1}) = y_{j-1}$ and $f(x_j) = y_j$. That is

$$A_j = \frac{y_j - y_{j-1}}{h_j} - \frac{h_j}{6}(M_j - M_{j-1}), \quad B_j = y_{j-1} - M_{j-1} \frac{h_j^2}{6}. \tag{2.60}$$

Now we will determine the constants M_j using the first derivative's continuity.

$$s_3'(x_j^-) = s_3'(x_j^+), \quad j = 1, \ldots, n-1. \tag{2.61}$$

That is equivalent to $j = 1, \ldots, n-1$,

$$\begin{aligned} s_3'(x_j^-) &= M_j \frac{h_j}{3} + M_{j-1} \frac{h_j}{6} + \frac{y_j - y_{j-1}}{h_j} \\ &= -M_j \frac{h_{j+1}}{3} - M_{j+1} \frac{h_{j+1}}{6} + \frac{y_{j+1} - y_j}{h_{j+1}} = s_3'(x_j^+). \end{aligned} \tag{2.62}$$

We can write the corresponding equations into a tri-diagonal linear system

$$\begin{pmatrix} \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & & & \\ & \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{h_{n-1}}{6} & \frac{h_{n-1}+h_n}{3} & \frac{h_n}{6} \end{pmatrix} \begin{pmatrix} M_0 \\ M_1 \\ \vdots \\ M_n \end{pmatrix} = \begin{pmatrix} \frac{y_2-y_1}{h_2} - \frac{y_1-y_0}{h_1} \\ \frac{y_3-y_2}{h_3} - \frac{y_2-y_1}{h_2} \\ \vdots \\ \frac{y_n-y_{n-1}}{h_n} - \frac{y_{n-1}-y_{n-2}}{h_{n-1}} \end{pmatrix} \tag{2.63}$$

In practice, the system will be rescaled for numerical stability.

$$\begin{pmatrix} \frac{h_1}{2(h_1+h_2)} & 1 & \frac{h_2}{2(h_1+h_2)} & & & \\ & \frac{h_2}{2(h_2+h_3)} & 1 & \frac{h_3}{2(h_2+h_3)} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{h_{n-1}}{2(h_{n-1}+h_n)} & 1 & \frac{h_n}{2(h_{n-1}+h_n)} \end{pmatrix} \begin{pmatrix} M_0 \\ M_1 \\ \vdots \\ M_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \end{pmatrix}$$

where $d_j = \frac{3}{h_{j-1}+h_j}\left[\frac{y_j-y_{j-1}}{h_j} - \frac{y_{j-1}-y_{j-2}}{h_{j-1}}\right]$. The above system still lacks 2 more constraints since the matrix is of size $(n-1)\times(n+1)$. Then we can apply the periodic spline or natural spline conditions. For example, if the natural constraint is applied: $s_3''(a) = s_3''(b) = 0$. We should have two more equations:

$$M_0 = M_n = 0. \tag{2.64}$$

Then we can simply ignore the first and last columns of the matrix (also M_0 and M_n). If the periodic constraint is imposed, then we can add two more constraints: $M_0 = M_n$ and

$$-M_0\frac{h_1}{3} - M_1\frac{h_1}{6} + \frac{y_1-y_0}{h_1} = M_n\frac{h_n}{3} + M_{n-1}\frac{h_n}{6} + \frac{y_n-y_{n-1}}{h_n}.$$

In both cases, the resulting linear system is still tri-diagonal and the solution takes $\mathcal{O}(n)$ time complexity with the Thomas algorithm.

Another popular choice to complete the matrix is to impose the constraints in the similar form on x_0 and x_n:

$$2M_0 + \frac{h_1}{h_0+h_1}M_1 = d_0, \quad \frac{h_n}{h_n+h_{n+1}}M_{n-1} + 2M_n = d_n \tag{2.65}$$

where $h_0 = h_{n+1} = 0$ and $d_0 = d_1$, $d_n = d_{n-1}$ are assumed.

Remark 2.3.2. *The Jacobi/Gauss-Seidel iteration also solves the system. It will converge to system accuracy within $\mathcal{O}(\log_2 \mathrm{u})$ iterations.*

2.3.3 Interpolation Error of Cubic Spline

The error estimate for cubic spline can be derived in a similar way as the Lagrange polynomial interpolation. The following result is attributed to Charles Hall (1968).

Theorem 2.3.3. *Let $f \in C^4([a,b])$ and $a = x_0 < \cdots < x_n = b$ is a set of nodes. Then the natural cubic spline s_3 interpolating f satisfies*

$$\|f - s_3\|_\infty \leq \frac{5}{384}\|f^{(4)}\|_\infty h^4 \tag{2.66}$$

where $h = \max_j |x_j - x_{j-1}|$.

Proof. Here we only state the rough idea to prove the error bound. Let $u(x)$ be the piecewise-defined Hermite interpolation polynomial that

$$u(x_j) = f(x_j), \quad u'(x_j) = f'(x_j). \tag{2.67}$$

then one can estimate

$$\max_{x\in[x_j,x_{j+1}]} |u - f| \leq \frac{1}{24}\|f^{(4)}\|_\infty (x - x_j)^2 (x - x_{j+1})^2 \leq \frac{1}{96}\|f^{(4)}\| h^4. \tag{2.68}$$

Then show that $\|s_3 - u\|_\infty$ is bounded by $\mathcal{O}(\|f^{(4)}\|_\infty h^4)$. This step needs to analyze the error $\|s_3' - f'\|_\infty$. We leave this as an exercise for the readers. □

2.4 Reproduce Kernel Hilbert Space

2.5 Exercises

The problems with $*$ tags are not required.

2.5.1 Theoretical Part

Problem 2.5.1. *Prove Corollary 2.1.13.*

Problem 2.5.2. *Estimate the interpolation error for e^x and $1/\sqrt{x}$ over the interval $[\frac{1}{2}, \frac{3}{2}]$ with equally spaced nodes and Chebyshev nodes, respectively.*

Problem 2.5.3. *Let $-1 \leq x_0 < x_1 < \cdots < x_n \leq 1$ be a set of interpolation nodes and $L_j(x)$ denotes the Lagrange basis polynomial at x_j. Prove if $x \in (x_i, x_{i+1})$, then*

$$L_i(x) + L_{i+1}(x) \geq 1.$$

Hint: Use Rolle's Theorem.

Problem 2.5.4. *Let the nodes $x_0, \dots, x_n \in [a,b]$ and $f \in C^{n+1}([a,b])$. For any given $x \in [a,b]$, prove there exists $\xi \in [a,b]$ such that the divided difference*

$$f[x_0, x_1, \dots, x_n, x] = \frac{1}{(n+1)!} f^{(n+1)}(\xi).$$

Hint: Construct the interpolation polynomial on nodes $x_0, \dots, x_n$ with Newton form first, then regard x as the additional node by (2.22), finally recall the Theorem 2.1.5.

Problem 2.5.5. *Show that the divided difference $f[x_0, x_1, \dots, x_n] = f[x_{\pi(0)}, x_{\pi(1)}, \dots, x_{\pi(n)}]$, where $\pi \in S_{n+1}$ is any permutation.*

Problem 2.5.6 (discrete circular convolution)**.** *Let two vectors $\mathbf{a} = (a_0, \dots, a_{N-1})$ and $\mathbf{b} = (b_0, \dots, b_{N-1})$ and we assume the convention that $a_{N+j} = a_j$ and $b_{N+j} = b_j$ to extend the vector to infinite size. The discrete circular convolution $\mathbf{c} = (c_0, \dots, c_{N-1}) = \mathbf{a} * \mathbf{b}$ is defined by*

$$c_j = \sum_{l=0}^{N-1} a_l b_{j-l}.$$

Prove that

$$\mathtt{DFT}(\mathbf{c}) = \mathtt{DFT}(\mathbf{a}) \circ \mathtt{DFT}(\mathbf{b}),$$

where $\circ$ is the Hadamard product or element-wise product.

Problem 2.5.7 (Minimum norm property)**.** *Let $f \in C^2([a,b])$ and s_3 be the natural cubic spline with interpolating f. Prove*

$$\int_a^b |s_3''(x)|^2 dx \le \int_a^b |f''(x)|^2 dx \tag{2.69}$$

The equality holds only when $f(x) = s_3(x)$ everywhere. Hint: One needs to prove

$$\int_a^b (f''(x) - s_3''(x)) s_3''(x) dx = 0, \tag{2.70}$$

then simply use the AM-GM inequality. The above equality can be proved using integration-by-parts twice (remember the interpolation conditions).

Problem 2.5.8. *Let the following tri-diagonal matrix (corresponding to natural cubic spline)*

$$A = \begin{pmatrix} 1 & \frac{h_2}{2(h_1+h_2)} & & & \\ \frac{h_2}{2(h_2+h_3)} & 1 & \frac{h_3}{2(h_2+h_3)} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{h_{n-2}}{2(h_{n-2}+h_{n-1})} \\ & & & \frac{h_{n-1}}{2(h_{n-1}+h_n)} & 1 \end{pmatrix}$$

and denote $B = A - I$. Prove $\|B\|_\infty \le \frac{1}{2}$ and $\|A^{-1}\|_\infty \le 2$.

Problem 2.5.9 $(*)$**.** *Complete the proof of Theorem 2.3.3.*

2.5.2 Computational Part

The coding part and test cases are available on GitHub.

Problem 2.5.10. *Implement the divided difference to compute the coefficients of the Newton form. You should call the Horner's scheme from the previous coding project. State the consequence when the length of* $[a, b]$ *is small.*

Problem 2.5.11. *Implement a routine to compute Hermite interpolation with* $m_j = 1$ *for each* $j = 0, \ldots, n$*. Estimate your routine's time complexity and space complexity.*

Problem 2.5.12. *Implement the cubic spline interpolation with periodic and natural spline conditions.*

Problem 2.5.13 $(*)$**.** *Implement discrete Fourier transform using a recursive approach (assume the total number of nodes is power of two).*

Chapter 3

Differentiation and Quadrature

Numerous applications such as calculation of tangent vectors or areas lead to the problem of computing

$$\mathcal{D}(f) := \frac{d}{dx}f(x), \quad \mathcal{I}(f) := \int_a^b f(x)dx, \tag{3.1}$$

for certain function $f(x) \in C^k([a,b])$. The accurate evaluations would be sometimes difficult if the analytic expression is absent. Especially when the function values of f are only accessible at a finite number of nodes. Therefore, it is important to find simple yet effective methods to approximate the derivatives and integrals.

3.1 Extrapolation

From the previous discussion, we already know that interpolation is to provide an estimate *within* the original observation range. The extrapolation is similar but aims to produce estimates outside the observation range. However, extrapolation may be subject to a greater uncertainty (Fig 3.1), one should use it only when overestimate is hardly occurring.

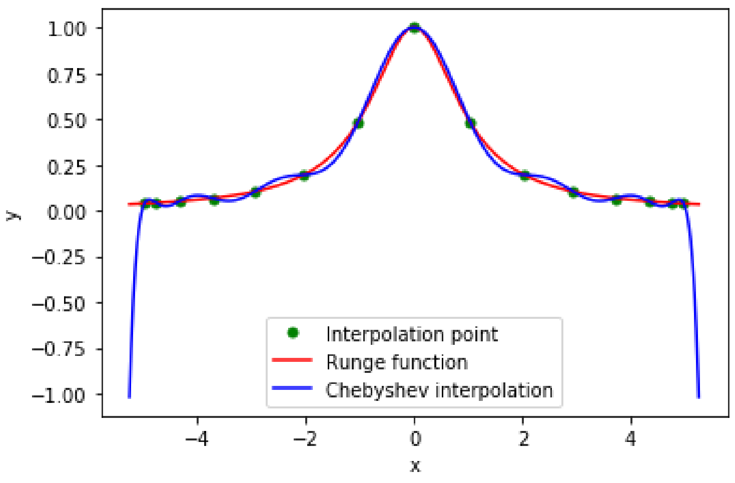

Figure 3.1: Extrapolation behavior for Chebyshev interpolation with 15 nodes.

3.1.1 Richardson Extrapolation

Suppose there is a sequence of estimates $A(h)$ depending the parameter h, the limit $A^* = \lim_{h\to 0^+} A(h)$ is the quantity to be computed. In practice, we only have access to $A(h)$ for a few values of h. Using these values to estimate A^* is a typical problem in extrapolation.

The basic idea is to use polynomial interpolation with a sequence of nodes $h_j \to 0$. Suppose the function $A(h)$ admits the following asymptotic expansion:

$$A(h) = a_0 + a_1 h^\gamma + a_2 h^{2\gamma} + \cdots + a_k h^{k\gamma} + \mathcal{O}(h^{(k+1)\gamma}) \tag{3.2}$$

for any $h > 0$ and $k \geq 0$. Then $A^* = a_0$ and $A(h) = A^* + \mathcal{O}(h^\gamma)$. Suppose we have access to the values $A(h_0), \dots, A(h_n)$, then this uniquely determines a polynomial $f_n \in \Pi_n$ and $f_n(h_j^\gamma) = A(h_j)$. We will approximate $A(0) \approx f_n(0)$. The computation of f_n follows the construction of the Newton form. The error estimate

Lemma 3.1.1. *Suppose h_j can be represented as*

$$h_j = \frac{\hbar}{t_j}$$

for some adjustable $\hbar$ and scaling constants $1 < t_0 < t_1 < \cdots < t_{n-1}$. Then

$$f_n(0) = A^* + (-1)^n \frac{a_{n+1}}{\prod_{j=0}^n t_j^\gamma} \hbar^{(n+1)\gamma} + \mathcal{O}(\hbar^{(n+2)\gamma}), \quad \text{as } \hbar \to 0.$$

Proof. We view $A(h)$ as a polynomial with respect to h^γ of degree $(n+1)$ with an addition perturbation $\mathcal{O}(h^{(n+2)\gamma})$. Then we have

$$A(h) = p_{n+1}(h^\gamma) + \mathcal{O}(h^{(n+2)\gamma}). \tag{3.3}$$

Let $\tilde{f}_n$ be the interpolation polynomial of degree n to p_{n+1},

$$p_{n+1}(x) \equiv \tilde{f}_n(x) + p[x, x_0, x_1, \dots, x_n] \prod_{j=0} (x - h_j^\gamma). \tag{3.4}$$

where $p[x, x_0, x_1, \dots, x_n]$ is the coefficient of p_{n+1}'s leading power, a_{n+1}. Thus

$$A^* = p_{n+1}(0) = \tilde{f}_n(0) + a_{n+1} \prod_{j=0} (0 - h_j^\gamma). \tag{3.5}$$

Use the result we have discussed in the stability of polynomial interpolation, see Chapter 2.1.5. Therefore

$$|\tilde{f}_n(0) - f_n(0)| \leq \lambda_n(0) \cdot \mathcal{O}(\hbar^{(n+2)\gamma}) \tag{3.6}$$

Here the Lebesgue function at zero $\lambda_n(0)$ is

$$\lambda_n(0) = \sum_{j=0}^n \prod_{k=0, k\neq j}^n \left| \frac{h_k^\gamma}{h_k^\gamma - h_j^\gamma} \right| = \sum_{j=0}^n \prod_{k=0, k\neq j}^n \left| \frac{1}{1 - (\frac{t_k}{t_j})^\gamma} \right|, \tag{3.7}$$

which is independent of $\hbar$. □

The Richardson extrapolation considers the special choice of $t_j = t^j$ for some $t > 1$. The error estimate then is

$$f_n(0) = A^* + \left(\frac{(-1)^n}{t^{n(n+1)\gamma/2}}a_{n+1}\right)\hbar^{(n+1)\gamma} + \mathcal{O}(\hbar^{(n+2)\gamma}). \tag{3.8}$$

In fact, there are easier ways to calculate the Richardson extrapolation. Consider the following

$$A(h) - t^\gamma A\left(\frac{h}{t}\right) = (1-t^\gamma)A^* + \cancel{a_1\left(h^\gamma - t^\gamma\left(\frac{h}{t}\right)^\gamma\right)} + a_2\left(h^{2\gamma} - t^\gamma\left(\frac{h}{t}\right)^{2\gamma}\right) + \dots \tag{3.9}$$

Let $A_1(h) = \frac{A(h)-t^\gamma A(\frac{h}{t})}{1-t^\gamma}$, we obtain the first iteration result as

$$A^* \approx A_1(h) + \mathcal{O}(h^{2\gamma}), \tag{3.10}$$

then follow the same idea, we cancel the $\mathcal{O}(h^{2\gamma})$ term by

$$A_1(h) - t^{2\gamma}A_1\left(\frac{h}{t^2}\right) = (1-t^{2\gamma})A^* + \mathcal{O}(h^{3\gamma}). \tag{3.11}$$

Therefore by taking $A_2(h) = \frac{A_1(h)-t^{2\gamma}A_1(\frac{h}{t^{2\gamma}})}{1-t^{2\gamma}}$, the second iteration satisfies

$$A^* \approx A_2(h) + \mathcal{O}(h^{3\gamma}). \tag{3.12}$$

However, it is not guaranteed that such process can constantly refine the approximation due to the potentially fast growing constant in the $\mathcal{O}$ notation.

3.1.2 Wynn's epsilon method

The Wynn's ε-method is another kind of extrapolation algorithm which is recommended as the best all-purpose acceleration method. It has a strong connection with Padé approximation and continued fractions. We will not cover the detailed derivation of the theory behind in this section. However, the Wynn's ε-method still has its limitation if the sequence converges to the desired value too slowly. The algorithm is stated as follows. Let $s_0, s_1, \dots, s_n, \dots$ be a sequence converging to the desired quantity,

1. Initialization. For $j = 0, 1, 2, \dots$, set

$$\varepsilon_{-1}^{(j)} = 0 \text{ (guarding elements)}, \quad \varepsilon_0^{(j)} = s_j.$$

2. Iteration. For $j, k = 0, 1, 2, \dots$,

$$\varepsilon_{k+1}^{(j)} = \varepsilon_{k-1}^{(j+1)} + [\varepsilon_k^{(j+1)} - \varepsilon_k^{j}]^{-1}.$$

The extrapolated results are stored at the columns $\varepsilon_{2l}^{(j)}$, $j, l = 0, 1, \dots$.

Example 3.1.2. *It is known that $\pi/4$ can be calculated by the asymptotic expansion:*

$$\arctan z = z - \frac{z^3}{3} + \frac{z^5}{5} - \dots \tag{3.13}$$

at $z = 1$. Define the function $A(h)$ such that

$$A(h) = \sum_{j=0}^{1/h} \frac{(-1)^j}{2j+1} = \frac{\pi}{4} + a_1 h + a_3 h^3 + \dots \tag{3.14}$$

Then we can approximation error is $\mathcal{O}(h)$, which is very slow. Taking $h = 10^{-3}$ has around 2×10^{-4} error.

1. *With Richardson Extrapolation, we can take $1/h = 250, 500, 1000, 2000$ and calculate the Richardson extrapolation three times would result with almost machine accuracy.*
2. *With Wynn's ε-method, we may take the sequence*

   $$s_k = \sum_{j=0}^{k} \frac{(-1)^j}{2j+1}$$

   as the truncated series at $z = 1$. With about 20 terms, we already reach the machine accuracy.

3.2 Differentiation with Finite Difference

Let $f \in C^2([a,b])$, we recall the Taylor expansion with reminder term,

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi), \tag{3.15}$$

where $\xi = \xi(x) \in [a,b]$, therefore we can compute the derivative by

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\xi). \tag{3.16}$$

This approximation offers a way to evaluate the derivative $f'(x)$ with error term $\mathcal{O}(h)$. Also the above formula is *exact* when f is polynomial of degree 1. We say an approximation has "degree k accuracy" if the approximation is *exact* for any polynomial of degree k.

Another important terminology is the "order". It describes the error term of the approximation. In the above case, the error term scales like $\mathcal{O}(h)$ as $h \to 0$, then the approximation is "of order 1" or "first order". In general, if the error term behaves like $\mathcal{O}(h^p)$, then we can say it is p-th order approximation.

The "stencil" refers to a set of nodes used in the approximation. In the above example, we have used $x, x+h$. We can of course create its sibling

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \frac{h}{2} f''(\zeta), \tag{3.17}$$

which uses the nodes $x - h, x$. When all nodes are $\geq x$ or $\leq x$, we say the scheme is forward or backward, respectively.

3.2.1 Finite Difference from Taylor Expansion

All results related to finite difference can be easily derived from Taylor expansion. Suppose we would like to approximate a high order derivative $f^{(m)}(x)$ with some nodes scattered around x in the following form.

$$\frac{1}{h^m}\sum_{j=0}^{n} c_j f(x+a_j h) = f^{(m)}(x) + E(x,h) \tag{3.18}$$

where E is the error term and $a_j \in \mathbb{Z}$ (sometimes half integers are used). Since $h \to 0$, we can expand all $f(x+a_j h)$ locally by Taylor series and truncate at least order $(m+1)$.

$$\frac{1}{h^m}\sum_{j=0}^{n} c_j \left(\sum_{k=0}^{p} \frac{1}{p!} f^{(p)}(x) a_j^p h^p + \frac{f^{(p+1)}(\xi_j)}{(p+1)!} a_j^{p+1} h^{p+1}\right) \tag{3.19}$$

Clearly, we need all lower (and maybe higher than m-th) order derivatives of f cancelled in the above summation, which is

$$\begin{aligned} &\sum_{j=0}^{n} c_j a_j^p = 0, \quad 0 \le p < m, \\ &\sum_{j=0}^{n} c_j a_j^m = m!, \end{aligned} \tag{3.20}$$

It is straightforward that $n \ge m$, otherwise the first equation system (Vandermonde matrix) must have zero solution. Suppose we have found a solution (c_j, a_j), $j = 0, \dots, n$, to the above system, then in the sequel, we try to estimate the error term $E(x,h)$. Especially, if $n = m$, here are two cases, let the following constant C

$$C = \sum_{j=0}^{n} c_j a_j^{m+1}, \quad C \text{ could be zero.}$$

1. If $C = 0$, then the error term can be estimated by expanding to $(m+2)$-th derivative.

$$E(x,h) = \frac{1}{h^m}\sum_{j=0}^{n} c_j \frac{f^{(m+2)}(\xi_j)}{(m+2)!} a_j^{m+2} h^{m+2} = h^2 \left(\sum_{j=0}^{n} c_j a_j^{m+2} \frac{f^{(m+2)}(\xi_j)}{(m+2)!}\right). \tag{3.21}$$

   One can expect higher order accuracy when more terms are involved.

2. If $C \neq 0$, then the error term is

$$E(x,h) = \frac{1}{h^m}\sum_{j=0}^{n} c_j \frac{f^{(m+1)}(\xi_j)}{(m+1)!} a_j^{m+1} h^{m+1} = h \left(\sum_{j=0}^{n} c_j a_j^{m+1} \frac{f^{(m+1)}(\xi_j)}{(m+1)!}\right). \tag{3.22}$$

Remark 3.2.1. *Here the ξ_j, $j = 0, \dots, n$ are in general distinct but possible to choose a single ξ to simplify the representation through the intermediate value theorem.*

Lemma 3.2.2. *If $f \in C^{m+1}(\mathcal{I})$, where $\xi_j \in \mathcal{I}$, then there exists $\xi \in \mathcal{I}$ such that*

$$\sum_{j=0}^{n} c_j a_j^{m+1} \frac{f^{(m+1)}(\xi_j)}{(m+1)!} = \sum_{j=0}^{n} c_j a_j^{m+1} \frac{f^{(m+1)}(\xi)}{(m+1)!}, \tag{3.23}$$

if $c_j a_j^{m+1} \geq 0$ (or ≤ 0), $j = 0, \dots, n$.

Proof. Define

$$\psi(x) = \sum_{j=0}^{n} c_j a_j^{m+1} \frac{f^{(m+1)}(x) - f^{(m+1)}(\xi_j)}{(m+1)!}$$

then $\max_j f(\xi_j) \geq 0$ and $\min_j f(\xi_j) \leq 0$. Then apply the intermediate value theorem. □

Example 3.2.3. *Let $n = 2, m = 2$ for an example. Then the equation system becomes*

$$\begin{aligned} c_0 + c_1 + c_2 &= 0, \\ c_0 a_0 + c_1 a_1 + c_2 a_2 &= 0, \\ c_0 a_0^2 + c_1 a_1^2 + c_2 a_2^2 &= 2. \end{aligned} \tag{3.24}$$

Then using Gauss elimination, one can find that

$$c_2(a_2 - a_0)(a_2 - a_1) = 2.$$

The above formula can be generalized. The constant $C = c_2(a_2 - a_0)(a_2 - a_1)(a_0 + a_1 + a_2)$. We list a few possible choices to satisfy (3.24).

1. *$(a_0, a_1, a_2) = (-1, 0, 1)$, $c_2 = 1$. Then $c_0 = 1$ and $c_1 = -2$ are derived. In this case $C = 0$. We will have the error term*

   $$E(x,h) = \frac{h^2}{24}\left(f^{(4)}(\xi_0) + f^{(4)}(\xi_2)\right) \underbrace{\Longrightarrow}_{3.2.2} \frac{h^2}{12} f^{(4)}(\xi).$$

   This is called central difference scheme.

2. *$(a_0, a_1, a_2) = (0, 1, 2)$, $c_2 = 1$. Then $c_1 = -2$, $c_0 = 1$, $C \neq 0$. The error is*

   $$E(x,h) = \frac{h}{6}(f^{(4)}(\xi_1) + 8f^{(4)}(\xi_2)) \underbrace{\Longrightarrow}_{3.2.2} \frac{3h}{2} f^{(4)}(\xi).$$

   This is the forward difference scheme.

The combination of the coefficients is not unique. The central scheme has better approximation due to its symmetry. In fact any combination satisfying $a_0 + a_1 + a_2 = 0$ should have the same order of error.

The general scheme with $a_j = j$ (or $-j$) can be derived from the following theorem.

Theorem 3.2.4. *In general, if $n = m$, then the forward difference scheme satisfies*

$$\begin{aligned}\sum_{j=0}^{n}(-1)^{n-j}\binom{n}{i}j^p &= 0, \quad 0 \le p < n \\ \sum_{j=0}^{n}(-1)^{n-j}\binom{n}{i}j^n &= n!.\end{aligned} \tag{3.25}$$

Proof. It is the easiest to prove by binomial transform. It can be also proved through the induction easily. Let

$$P_0(x) = (x-1)^n, \quad P_k(x) = xP'_{k-1}(x),$$

then one can show inductively that P_k, $k \ge 1$, has following form

$$P_k(x) = n(n-1)\cdots(n-k+1)(x-1)^{n-k}x^k + (x-1)^{n-k+1}F(x) \tag{3.26}$$

with $F(x)$ as a polynomial of at most degree $k-1$. This can be easily proved since

$$\begin{aligned}P_{k+1} = xP'_k(x) = n(n-1)\cdots(n-k)(x-1)^{n-k-1}x^{k+1} &+ \\ + (x-1)^{n-k}n(n-1)\cdots(n-k+1)kx^k& \\ + (x-1)^{n-k}(n-k+1)F(x)& \\ + (x-1)^{n-k}(x-1)F'(x).&\end{aligned} \tag{3.27}$$

It is clear the last three terms can merge into the form as (3.26). Therefore $P_k(1) = 0$ unless $k = n$ and $P_n(1) = n!$ is immediately obtained.

Now if we expand the polynomial P_0 as monomial,

$$P_0(x) = \sum_{j=0}^{n}\binom{n}{j}x^j(-1)^{n-j}, \tag{3.28}$$

then it is not difficult to show that

$$P_k(x) = \sum_{j=0}^{n}\binom{n}{j}j^k x^j(-1)^{n-j}$$

through induction as well, which is exactly our conclusion by setting $x = 1$. □

Theorem 3.2.5 (forward difference).

$$f^{(n)}(x) = \frac{1}{h^n}\sum_{j=0}^{n}(-1)^{n-j}\binom{n}{j}f(x+jh) + \mathcal{O}(h).$$

The corresponding schemes of backward difference and central difference can be derived similarly (see the exercise).

3.2.2 Rounding Error Issue

The finite difference formula provides a simple and effective way to evaluate the derivatives, however its formulation would be sensitive to rounding errors. Take the central difference scheme for $f''(x)$ as an example, one can derive a similar estimate for higher order derivatives.

Example 3.2.6.

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \frac{h^2}{12} f^{(4)}(\xi) \tag{3.29}$$

The error comes from two sources. The truncation error term from $\frac{h^2}{12} f^{(4)}(\xi)$ and the rounding error from the evaluation of the first term by the basic arithmetic operations. Suppose the addition/subtraction are implemented by Kahan sum (see exercises of Chapter 1) which almost does not introduce errors in the arithmetic operations. Then the rounding error of $f(x+h) - 2f(x) + f(x-h)$ is at most $4\max_{x\in\mathcal{I}} |f(x)|\mathrm{u}$. Therefore the total error

$$|E_{total}| \le \frac{4\max_{x\in\mathcal{I}} |f(x)|\mathrm{u}}{h^2} + \frac{h^2}{12} \max_{x\in\mathcal{I}} |f^{(4)}(x)| \tag{3.30}$$

Minimizing the right-hand-side of (3.30), let $M = \max_{x\in\mathcal{I}} |f(x)| \max_{x\in\mathcal{I}} |f^{(4)}(x)|$, we obtain

$$\min_{h\in\mathbb{R}} |E_{total}| \le \sqrt{\frac{4}{3}\mathrm{u}M}.$$

The optimal achieves at $h^* = \sqrt[4]{48\mathrm{u}M}$. For example, if $f(x) = \exp(x)$ and evaluate its second derivative around $x = 0$, then $\mathcal{M} \sim 1$, the error is around 1.3×10^{-8} for $h^* \sim 2.5 \times 10^{-4}$.

For higher order derivatives, the rounding error would have even greater impact for the finite difference schemes. Then it is much more important to avoid h being too small.

3.2.3 Improve by Extrapolation

Now we can combine the previous discussed extrapolation technique to acquire higher ordered scheme. We use a very simple example to show how this works.

Example 3.2.7. *Suppose $A(f, h)$ is the central difference scheme for $f'(x)$, which is*

$$A(f, h) = \frac{f(x+h) - f(x-h)}{2h} \tag{3.31}$$

the previous discussion has claimed that $A(f, h) = f'(x) + \mathcal{O}(h^2)$. Now we try to fit the formulation in the framework of extrapolation. Formally, we can expand $f(x \pm h)$ with Taylor series with infinite terms (might not converge though), that is,

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \cdots \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + \cdots \end{aligned} \tag{3.32}$$

Therefore $A(f,h) = f'(x) + \frac{h^2}{6}f''(x) + \frac{h^4}{120}f''''(x) + \cdots$. Here the coefficients are all formal since the convergence is not guaranteed. In the next, we take $A(f,\frac{h}{2})$, which uses a smaller step length to approximate $f'(x)$, then

$$A(f,h/2) = f'(x) + \frac{h^2}{24}f''(x) + \frac{h^4}{1920}f''''(x) + \cdots \tag{3.33}$$

Cancel the $\mathcal{O}(h^2)$ term by

$$\frac{1}{3}\left(4A(f,h/2) - A(f,h)\right) = f'(x) - \frac{h^4}{480}f''''(x) + \cdots .$$

In this way we have built a more accurate formula $A_1(f,h) = \frac{4A(f,h/2)-A(f,h)}{3}$ for $f'(x)$, the error is fourth order. Bring the definition of the finite difference scheme into A_1, then

$$\begin{aligned} A_1(f,h) &= \frac{4}{3}\left(\frac{f(x+h/2)-f(x-h/2)}{h}\right) - \frac{1}{3}\left(\frac{f(x+h)-f(x-h)}{2h}\right) \\ &= \frac{-f(x+h)+8f(x+h/2)-8f(x-h/2)+f(x-h)}{6h}. \end{aligned} \tag{3.34}$$

This central difference scheme has $\mathcal{O}(h^4)$ error.

The above example can still iterate through the extrapolation process, since $A_1(f,h) = f'(x) + \mathcal{O}(h^4)$, we can use $A_2(f,h) = \frac{16}{15}A_1(f,h/2) - \frac{1}{15}A_1(f,h)$ to cancel out the $\mathcal{O}(h^4)$ term which leads to a $\mathcal{O}(h^6)$ error. However one should also notice this process require more nodes for computation: A_1 needs nodes $x \pm h$, $x \pm h/2$, A_2 will acquire additional nodes $x \pm h/4$ for evaluation. Such higher precision evaluation method takes more computational time, sometimes we need to trade off the efficiency and accuracy.

Remark 3.2.8. *One of the advantage of using extrapolation is that h does not have to be too small which is sensitive to numerical rounding errors. The potential issue would be growth of derivative with respect to order, for sufficiently smooth functions, extrapolation usually produces quite accurate evaluations. The potential limitation of the Richardson extrapolation is the requirement of known asymptotic expansion (formally only), while the Wynn-ε method does not have such limitation.*

3.3 Quadrature Rules

The numerical quadrature find the value of an integral

$$\mathcal{I}(f) = \int_a^b f(x)dx$$

from the function values at a finite number of points. We are mostly interested in the following quadrature formula

$$\mathcal{I}_n(f) = (b-a)\sum_{j=0}^{n} w_j f(x_j) \tag{3.35}$$

where x_j are the nodes and w_j are the weights. Similar to the numerical derivatives, we also define some terminologies. The formula $\mathcal{I}_n$ is said to have degree k accuracy if $\mathcal{I}_n$ is exact for all polynomials $f \in \Pi_k$. Since the integration formula is linear, the exactness can be rephrased as

$$\begin{aligned} \mathcal{I}(x^p) &= \mathcal{I}_n(x^p), \quad 0 \le p \le k, \\ \mathcal{I}(x^{k+1}) &\neq \mathcal{I}_n(x^{k+1}). \end{aligned} \tag{3.36}$$

3.3.1 Interpolation Based Rules

The interpolation based idea is intuitive. Let $q_n(x)$ be the interpolation polynomial on the nodes x_j with values $f(x_j)$, $j = 0, 1, \dots, n$, respectively. We define the quadrature by interpolation formula as

$$\mathcal{I}_n(f) := \int_a^b q_n(x)dx \tag{3.37}$$

The above quadrature formula is exact for all degree n polynomials f, therefore it has <u>at least</u> degree n accuracy.

Remark 3.3.1. *In the section of interpolation, we have seen that the L^∞ error between f and q_n could be large (e.g. Runge phenomenon) as $n \to \infty$. So the nodes would be important as well for quadratures.*

Using the Lagrange polynomials, we can represent

$$q_n(x) = \sum_{j=0}^n f(x_j)L_j(x), \quad L_j(x) = \prod_{k=0, k\neq j}^n \frac{x - x_k}{x_j - x_k}. \tag{3.38}$$

Then it is not difficult to derive

$$\begin{aligned} \mathcal{I}_n(f) &= \sum_{j=0}^n f(x_j) \int_a^b L_j(x)dx \\ &= (b-a)\sum_{j=0}^n f(x_j) \int_0^1 \prod_{k=0,k\neq j}^n \frac{t - t_k}{t_j - t_k}dt, \quad t_j = \frac{x_j - a}{b - a}. \end{aligned} \tag{3.39}$$

therefore the weights $w_j = \int_0^1 \prod_{k=0,k\neq j}^n \frac{t-t_k}{t_j-t_k}dt$.

Example 3.3.2 (rectangle rule). *The rectangle rule is the simplest one, where we choose $x_0 = \frac{a+b}{2}$ as the middle point. Then the quadrature rule writes*

$$\mathcal{I}_{0,rectangle}(f) = (b-a)f(x_0).$$

Such rule is exact for any linear functions, therefore it has degree 1 accuracy. We can see that the degree of exactness could exceed n. Later we will see the maximum degree of exactness for such form is $2n+1$ in the next chapter.

Example 3.3.3 (trapezoid rule). *The trapezoid rule takes $x_0 = a$ and $x_1 = b$.*

$$\mathcal{I}_{1,trapezoid}(f) = (b-a)\left(\frac{1}{2}f(x_0) + \frac{1}{2}f(x_1)\right).$$

One can check that this rule is exact for $f(x) = 1, x$. It is also having degree 1 accuracy. It actually has a slightly larger constant in error estimate than the rectangle rule.

3.3.2 Numerical Error of Interpolation Based Rules

Now we try to estimate $|\mathcal{I}(f) - \mathcal{I}_n(f)|$ from above derivation.

Theorem 3.3.4. *Suppose the quadrature rule $\mathcal{I}_n$ has at least degree r accuracy that $r \geq n$ and $f \in C^{r+1}([a,b])$. Then*

$$|\mathcal{I}(f) - \mathcal{I}_n(f)| \leq \Omega_r \frac{(b-a)^{r+2}}{(r+1)!} \max_{x\in[a,b]} |f^{(r+1)}(x)|, \tag{3.40}$$

where the constant Ω_k is defined by

$$\Omega_r := \min_{t_{n+1},\dots,t_r \in [0,1]} \int_0^1 \prod_{j=0}^{r} |t - t_j| dt, \quad t_j = \frac{x_j - a}{b-a}, j = 0, \dots, n. \tag{3.41}$$

Proof. Let $x_{n+1}, \dots, x_r$ be additional distinct nodes on $[a,b]$ and define f_r the interpolating polynomial on the nodes $x_0, \dots, x_r$, since the quadrature rule has degree r accuracy, then

$$\mathcal{I}_n(f) = (b-a)\sum_{j=0}^{n} w_j f(x_j) = (b-a)\sum_{j=0}^{n} w_j f_r(x_j) = \mathcal{I}_n(f_r) = \mathcal{I}(f_r).$$

Using the theories developed in Interpolation, we know that

$$f(x) - f_r(x) = \frac{\omega_r(x) f^{(r+1)}(\xi)}{(r+1)!}, \tag{3.42}$$

where $\omega_r(x) = \prod_{j=0}^{r}(x - x_j)$, therefore

$$|\mathcal{I}(f - f_r)| = \left| \int_a^b \frac{\omega_r(x) f^{(r+1)}(\xi)}{(r+1)!} dx \right| \leq \frac{(b-a)}{(r+1)!} \left(\int_a^b |\omega_r(x)| dx \right) \max_{x\in[a,b]} |f^{(r+1)}(x)|. \tag{3.43}$$

Since $x_{n+1}, \dots, x_r$ can be chosen arbitrarily, then we select the combination that minimizes $\left(\int_a^b |\omega_r(x)| dx\right)$, which will lead to our conclusion by a simple scaling. □

Remark 3.3.5. *As we can see, the error from interpolation based quadrature has a similar form of the interpolation polynomial's error. This implies the Runge phenomenon would occur as well. In fact, the Runge function's integration will not converge on uniformly distributed nodes.*

One way to overcome the issue of Runge phenomenon is to perform piecewise integration. Suppose $[a,b]$ is divided into N sub-intervals of equal sizes, each has length $H = \frac{b-a}{N}$. Then the quadrature error on each sub-interval would be:

$$\Omega_r \frac{H^{r+2}}{(r+1)!} \max_{x\in[a,b]} |f^{(r+1)}(x)|$$

where Ω_r is independent of the interval length. Therefore the total quadrature error would be bounded by

$$N\Omega_r \frac{H^{r+2}}{(r+1)!} \max_{x\in[a,b]} |f^{(r+1)}(x)| = (b-a)\Omega_r \frac{H^{r+1}}{(r+1)!} \max_{x\in[a,b]} |f^{(r+1)}(x)| = \mathcal{O}((b-a)H^{r+1}).$$

Example 3.3.6 (rectangle rule). *For rectangle rule, $r = 1, n = 0$, therefore*

$$\Omega_r = \min_{t_1} \int_0^1 |t - \frac{1}{2}||t - t_1|dt = \frac{1}{12}, \quad (t_1 = \frac{1}{2}). \tag{3.44}$$

This is easiest to notice by changing variable $s = t - \frac{1}{2}$ and the symmetry, then the integral is just

$$\Omega_r = \min_{z\in[-1/2,1/2]} \int_{-1/2}^{1/2} |s||s - z|ds = \min_{z\in[0,1/2]} \int_0^{1/2} s(|s - z| + |s + z|)ds \geq \int_0^{1/2} s(2s)ds.$$

Example 3.3.7 (trapezoid rule). *For trapezoid rule, $r = n = 1$, therefore*

$$\Omega_r = \int_0^1 (1 - t)tdt = \frac{1}{6}. \tag{3.45}$$

then the error is bounded by $\frac{(b-a)h^2}{12} \max_{x\in[a,b]} |f^{(r+1)}(x)|$.

3.3.3 Newton-Cotes Formula

The Newton-Cotes formula is a special interpolation based quadrature rule. The nodes are equally spaced. The rectangle and trapezoid rules are just two simplest cases. We define

1. closed form, $x_0 = a$, $x_n = b$, $x_j = a + jh$, $h = \frac{b-a}{h}$, $n \geq 1$.
2. open form, $x_0 = a + h$, $x_n = b - h$, $h = \frac{b-a}{n+2}$, $n \geq 0$.

The difference is whether the end points are included or not. Using the previous result, we can compute the quadrature weights by

$$\begin{aligned} w_j &= \int_0^1 \prod_{k=0,k\neq j}^n \frac{t - t_k}{t_j - t_k} dt = \int_0^1 \prod_{k=0,k\neq j}^n \frac{nt - k}{j - k} dt \\ &= \frac{1}{n} \int_0^n \prod_{k=0,k\neq j}^n \frac{s - k}{j - k} ds. \end{aligned} \tag{3.46}$$

The computations of the weights can be efficient by noticing the symmetry.

Lemma 3.3.8. $w_j = w_{n-j}$.

Proof. This is because $w_j = \int_a^b L_j(x)dx = \int_a^b L_j(a + b - x)dx = \int_a^b L_{n-j}(x)dx = w_{n-j}$. □

The weights w_j are only relevant to n and j. In practice, these values are tabulated *a priori*. When $n \geq 2$, the weights include negative terms for open forms, and when $n \geq 8$, the weights include negative terms for closed forms, which could introduce numerical instability from rounding errors. Therefore one should only limit to small values of n.

Table 3.1: A few examples for closed form

n	w_j	r	M_n
1	$(1/2, 1/2)$	1	$1/12$
2	$(1/6, 2/3, 1/6)$	3	$1/90$
3	$(1/8, 3/8, 3/8, 1/8)$	3	$3/80$
4	$(7/90, 32/90, 12/90, 32/90, 7/90)$	5	$8/945$

Remark 3.3.9. *We can derive the error estimate for Newton-Cotes formula using the result from last section.*

$$\begin{aligned}|\mathcal{I}(f) - \mathcal{I}_{n,NC}(f)| &\le \Omega_r \frac{(b-a)^{r+2}}{(r+1)!} \\ &= M_n h^{r+2} \max_{x\in[a,b]} |f^{(r+1)}(x)|\end{aligned} \tag{3.47}$$

where $M_n = \Omega_r n^{r+2} \frac{1}{(r+1)!}$, *see the following table for a reference.*

From the above table, we can notice that when n is even, the exactness is at $r = n+1$ for closed forms. This actually is a general statement.

Lemma 3.3.10. *For* $n \ge 2$ *even, the closed forms of Newton-Cotes formula have degree* $r = n+1$ *accuracy.*

Proof. First, we know that $r \ge n$. Consider any polynomial of degree $n+1$,

$$p(x) = \sum_{j=0}^{n+1} b_j x^j, \tag{3.48}$$

we can rewrite the polynomial by

$$p(x) = b_{n+1}(x - \frac{a+b}{2})^{n+1} + \sum_{j=0}^{n} b'_j x^j \tag{3.49}$$

with another set of coefficients b'_j. The first term will have the integral as zero on the interval $[a, b]$. Numerically, using Newton-Cotes formula,

$$\mathcal{I}_{n,NC}((x - \frac{a+b}{2})^{n+1}) = (b-a)\sum_{j=0}^{n} w_j (x_j - \frac{a+b}{2})^{n+1} \tag{3.50}$$

while $x_{n-j} - \frac{a+b}{2} = -(x_j - \frac{a+b}{2})$ and $w_j = w_{n-j}$, we can cancel all terms.

It still remains to show that $\mathcal{I}(x^{n+2}) \ne \mathcal{I}_{n,NC}(x^{n+2})$. Because $r = (n+1)$ degree of accuracy is achievable, then we borrow the previous estimate result, let $f(x) = x^{n+2}$,

$$\begin{aligned}\mathcal{I}(f) - \mathcal{I}_n(f) &= \int_a^b \frac{\omega_{n+1}(x) f^{(n+2)}(\xi)}{(n+2)!} dx = \int_a^b \omega_{n+1}(x) dx \\ &= \int_a^b \omega_n(x)(x - x_{n+1}) dx = F(x)(x - x_{n+1})|_a^b - \int_a^b F(x) dx\end{aligned} \tag{3.51}$$

where $F(x)$ is defined by

$$F(x) := \int_a^x \omega_n(t)dt.$$

Then it is simple to derive that $F(a) = F(b) = 0$ using the symmetry. Now we only have to show that

$$\int_a^b F(x)dx \neq 0.$$

In fact, we can show a stronger claim: $F(x) > 0$ over (a, b). This is left as an exercise. □

Since the Newton-Cotes formula definitely will fail on evaluating the integral of Runge function $f(x) = \frac{1}{1+x^2}$ on the interval $[-5, 5]$. It is more practical to combine the piecewise integral technique, which is called composite Newton-Cotes formula. In the following, we discretize the interval $[a, b]$ into m sub-intervals of the same size $H = \frac{b-a}{m}$, then on each sub-interval, we apply the Newton-Cotes formula (say closed form) with $(n + 1)$ equally spaced nodes. Then the numerical integral would have an error bounded by

$$mM_n \left(\frac{H}{n}\right)^{r+2} \max_{x\in[a,b]} |f^{(r+1)}(x)| = (b-a)\frac{M_n}{n}\left(\frac{H}{n}\right)^{r+1} \max_{x\in[a,b]} |f^{(r+1)}(x)| \tag{3.52}$$

where $r = n$ for odd n and $r = n + 1$ for even n, see last section for a quick derivation.

Example 3.3.11 (composite trapezoid rule). *The composite trapezoid rule is often used for practical integration especially when f is periodic. Let $x_j = a + jH$, $j = 0, \dots, m$,*

$$T(f, H) = \frac{H}{2}\left(f(a) + 2\sum_{j=1}^{m-1} f(x_j) + f(b)\right).$$

Its error then can be estimated by

$$\frac{1}{12}(b-a)H^2 \max_{x\in[a,b]} |f''(x)| = \mathcal{O}((b-a)H^2). \tag{3.53}$$

In the next, we take a more careful look at the composite trapezoid rule. Recall the asymptotic Euler-Maclaurin summation formula:

$$\sum_{j=0}^{m} g(j) \sim \int_0^m g(x)dx + \frac{g(0) + g(m)}{2} + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!}(g^{2k-1}(m) - g^{(2k-1)}(0)) \tag{3.54}$$

If we take $g(j) = f(a + jH)$, then we will arrive at

$$T(f, H) \sim \int_a^b f(x)dx + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!} H^{2k}\left(f^{(2k-1)}(b) - f^{(2k-1)}(a)\right) \tag{3.55}$$

which means there is an asymptotic expansion in the form of

$$T(f, h) = \int_a^b f(x)dx + c_2H^2 + c_4H^4 + \cdots. \tag{3.56}$$

Particularly, for smooth periodic function, the Euler-Maclaurin summation *formally* shows the numerical error is less than any polynomial of H.

3.3.4 Romberg Integration

The composite trapezoid rule's asymptotic expansion (3.56) implies a Richardson extrapolation combination to accelerate the evaluation. The Romberg integration refers to the following scheme:

1. Compute the sequence $a_{l,0} = T(f, (b-a)/2^l)$, $l = 0, \dots, L$, for the standard composite trapezoid rule with different sub-interval sizes.
2. Extrapolation by
$$a_{l,q+1} = \frac{4^{q+1}a_{l,q} - a_{l-1,q}}{4^{q+1}-1}, \quad q = 0, \dots, L-1 \text{ and } l = q+1, \dots, L$$
3. Output $a_{L,L}$, which should have an error of $\mathcal{O}(H^{2L+2})$, $H = (b-a)/2^L$.

Remark 3.3.12. *One of advantage of the Romberg method is the re-use of the nodes. This is extremely helpful when evaluating f is not cheap. The extrapolation process also builds a new quadrature formulation implicitly. This quadrature rule brings the error $\mathcal{O}(n^{-2})$ to $\mathcal{O}(n^{-2\log_2 n - 2})$, where n is the total number of nodes. Although the computational time increases a few times, the return seems worth it when f is sufficiently smooth.*

3.3.5 Adaptive Integrations

Numerically, we can apply any composite quadrature rule on a successive partition of $[a, b]$ until the estimated error is under tolerance. Let $A(f, H)$ denote any composite quadrature rule (e.g. Newton-Cotes), $H = (b-a)/m$, then

$$A(f, H) = \int_a^b f(x)dx + \mathcal{O}(H^{r+1}) \tag{3.57}$$

where r is the degree of accuracy. Then $A(f, H/2)$ will presumably introduce an error of about $2^{-(r+1)}$ times the size of the previous case. Therefore we obtain an rough estimate of the error by
$$\mathcal{E} \approx \left| \frac{A(f, H) - A(f, H/2)}{1 - 2^{-(r+1)}} \right|$$
One can successively halve H until the estimated error is less than tolerance. However such method is not efficient when the quadrature on most of the sub-intervals are already very accurate. In this case, the best strategy is to keep those accurate sub-intervals and only partition the rest. This process will produce a non-uniform distribution of sub-intervals.

There are several ways of implementation. The simplest recursive algorithm can be roughly described as follows. Let $A(f, \alpha, \beta)$ be any quadrature rule on $[\alpha, \beta]$ with degree of accuracy r and $\mathcal{E}(f, \alpha, \beta)$ be an estimate of error for $|A(f, \alpha, \beta) - \int_\alpha^\beta f(x)dx|$. Then

1. Initially, $\alpha = a$, $\beta = b$, ε is the tolerance, $\mathcal{I} = 0$.
2. If $|\mathcal{E}(f, \alpha, \beta)| \le \varepsilon \frac{\beta - \alpha}{b - a}$ or $|\beta - \alpha|$ is too small, $\mathcal{I} = \mathcal{I} + A(f, \alpha, \beta)$, stop. Otherwise goto step 3.

3. Divide $[\alpha, \beta]$ into $[\alpha, \gamma]$, $[\gamma, \beta]$, $\gamma = \frac{\alpha+\beta}{2}$.
   (a) For the 1st half, let $\alpha = \alpha$, $\beta = \gamma$, goto step 2.
   (b) For the 2nd half, let $\alpha = \gamma$, $\beta = \beta$, goto step 2.

Ideally, the automatic partition will generate a non-uniform distributed sub-intervals. The total estimated numerical error will be bounded by ε.

A slightly more economical plan is stated in the section 9.7 of the book **?**, which is focusing on the left-most sub-interval until its error bound is under some tolerance, then eliminate the chosen sub-interval and restart with the rest.

3.3.6 Improper Integral

The jump discontinuity is in general simple to deal with. We will focus on the unbounded functions, for example, $\log x$, x^γ with $0 > \gamma > -1$. These singularities are integrable. Let us take the following function for an example:

$$f(x) = \phi(x)(x-a)^\gamma, \quad x \in [a, b].$$

where $\phi(x)$ is a smooth function. Using integration by parts,

$$\begin{aligned}\int_a^b \phi(x)(x-a)^\gamma dx &= \frac{(x-a)^{\gamma+1}}{\gamma+1}\phi(x)\Big|_a^b - \int_a^b \phi'(x)\frac{(x-a)^{\gamma+1}}{\gamma+1}dx \\ &= \cdots \\ &= \frac{(x-a)^{\gamma+1}}{\gamma+1}\phi(x)\Big|_a^b - \frac{(x-a)^{\gamma+2}}{(\gamma+1)(\gamma+2)}\phi'(x)\Big|_a^b + \cdots \\ &+ \int_a^b \phi^{(p)}(x)\frac{(-1)^p(x-a)^{\gamma+p}}{(\gamma+1)(\gamma+2)\cdots(\gamma+p)}dx,\end{aligned} \tag{3.58}$$

where the last integral is sufficiently regular and can be evaluated by quadrature rules such as Newton-Cotes formula, the error can be estimated using the result developed in previous sections. The first p terms are explicitly known. There are other choices such as series solution, which is

$$\int_a^b \frac{\phi(x)}{(x-a)^\gamma}dx = \int_a^b \sum_{k=0}^{p} \frac{\phi^{(k)}(a)(x-a)^{k-\gamma}}{k!}dx + \int_a^b \frac{\phi^{(p+1)}(\xi)(x-a)^{p+1-\gamma}}{(p+1)!}dx \tag{3.59}$$

The reminder determines the convergence of the series. As long as the reminder is decaying, one may use extrapolation technique to find the integral without using too many iterations. Otherwise the integral part has to be evaluated through quadrature rules.

Another typical method is to isolated the singularity

$$\int_a^b \frac{\phi(x)}{(x-a)^\gamma}dx = \int_a^{a+\varepsilon} \frac{\phi(x)}{(x-a)^\gamma}dx + \int_{a+\varepsilon}^b \frac{\phi(x)}{(x-a)^\gamma}dx \tag{3.60}$$

where the integral around singularity might converge when ε is sufficiently small (less than convergence radius of Taylor series). The non-singular integral can be computed by the traditional methods, since the integrand grows fast when $\varepsilon \to 0$, it is more suitable to apply adaptive methods.

Another class of improper integral is with infinite domain. In general, this problem is difficult (e.g. oscillatory integral), we only discuss the simplest case. Let $f(x)$ be integrable over $[a, \infty)$ and assume there exists $s > 0$ such that

$$\lim_{x\to\infty} x^{1+s} f(x) = 0. \tag{3.61}$$

The most intuitive way is to truncate the interval $[a, \infty)$ to $[a, c]$ such that the integral on $[c, \infty)$ is negligible. The point c sometimes can be inferred from *a priori* estimate, sometimes has to be found dynamically. The other methods are more or less playing around the change of variable to make the interval "finite" (e.g. $x \mapsto x^{-\beta}$ when $x \ge c > 0$).

Other advanced tools (e.g. complex analysis) and examples will be discussed in later topic chapter and integral equation chapter.

3.4 Gauss Quadrature

The Gauss quadrature maximizes the exactness of quadrature rules. Let $x_0, \dots, x_n$ the nodes on $[-1, 1]$, in the following we will discuss the numerical quadrature for the weighted integral

$$\mathcal{I}_w(f) = \int_{-1}^{1} w(x) f(x) dx \simeq \mathcal{I}_{n,w}(f) := \sum_{j=0}^{n} c_j f(x_j),$$

where the coefficients are determined later. We can also borrow the same accuracy concept from previous chapter ($w(x) = 1$). It is clear that with $n + 1$ nodes, the accuracy is at least degree n, in the later derivation we will see that Gauss quadrature has an accuracy of $r = n+m$ for $m > 0$.

Theorem 3.4.1. *Let* $\omega(x) = \prod_{j=0}^{n}(x - x_j)$*, then*

$$\langle \omega(x), p_k \rangle_w = 0, \quad 0 \le k \le m - 1$$

if and only if the associated quadrature rule has accuracy of $n + m$.

Proof. The key idea is represent any polynomial q of Π_{n+m} by

$$q(x) = \omega(x)s(x) + t(x) \tag{3.62}$$

where $s(x) \in \Pi_{m-1}$ and $t \in \Pi_n$. Therefore the quadrature over the reminder term $t(x)$ is exact,

$$\sum_{j=0}^{n} c_j t(x_j) = \int_{-1}^{1} t(x)w(x)dx = \int_{-1}^{1} q(x)w(x)dx - \underline{\int_{-1}^{1} \omega(x)s(x)w(x)dx}. \tag{3.63}$$

□

It is clear that the maximum of $m \le n+1$, otherwise we can choose $s(x) = \omega(x)$, then $\langle \omega(x), \omega(x)\rangle_w > 0$ violates the above theorem.

Corollary 3.4.2. *The quadrature rule*

$$\mathcal{I}_{n,w}(f) = \sum_{j=0}^{n} c_j f(x_j)$$

has maximum accuracy of degree $2n+1$.

The next question would be whether the maximum $2n+1$ is achievable. This requires that

$$\int_{-1}^{1} \omega(x) p_k(x) w(x) dx = 0, \quad 0 \le k \le n. \tag{3.64}$$

This formula indicates that $\omega(x) = p_{n+1}(x)$.

Theorem 3.4.3. *The only choice of ω satisfying (3.64) is p_{n+1}.*

Proof. Without loss of generality, we assume p_{n+1}'s leading power's coefficient is one (for example, the recursive formula (4.22)). Since p_{n+1} also satisfies the orthogonal relation. Therefore

$$\int_{-1}^{1} (\omega(x) - p_{n+1}(x)) s(x) w(x) dx = 0, \quad s \in \Pi_n \tag{3.65}$$

while $\omega - p_{n+1} \in \Pi_n$, then we have a contradiction if we take $s(x) = \omega - p_{n+1} \neq 0$. □

The above theorem implies that the nodes x_j are the zeros of p_{n+1} (we still need to show they are simple roots). The corresponding quadrature rule is called Gauss quadrature, the accuracy is of degree $2n+1$.

For $w = 1$, the Legendre polynomial's roots are not at the end points, while sometimes the end points ± 1 are useful to be included in the quadrature nodes, therefore we may want to generate a similar Gauss quadrature with the end nodes as well. Following the same idea, in order to make ± 1 as the roots of $\omega(x)$ but also keep ω concentrated on p_k for large k, we can define

$$\bar{\omega}(x) := p_{n+1}(x) + A p_n(x) + B p_{n-1}(x) \tag{3.66}$$

the constants A, B are used to control $\bar{\omega}(\pm 1) = 0$. The corresponding $\bar{\omega}$ has the similar property as ω but only provides an accuracy of degree $2n-1$. The nodes are roots of $\bar{\omega}$, they are called Gauss-Lobatto nodes. In the following, we prove some of the properties of Gauss quadrature.

Theorem 3.4.4. *All roots x_j of p_{n+1} are distinct.*

Proof. Pick out all roots that are having odd multiplicity, say $z_0 < z_1 < \cdots < z_M$, then

$$q(x) = (x - z_0) \cdots (x - z_M) \tag{3.67}$$

should satisfy that $q(x) p_{n+1}(x) w(x) > 0$ except on the roots. However if $M \neq n$, we would have an issue since

$$\int_{-1}^{1} q(x) p_{n+1}(x) w(x) dx = 0. \tag{3.68}$$

□

Theorem 3.4.5. *The weights c_j for Gauss quadrature rules*

$$\mathcal{I}_n(f) = \sum_{j=0}^{n} c_j f(x_j) \tag{3.69}$$

are all positive.

Proof. The polynomials p_k satisfies the recursive formula

$$p_{k+1}(x) = (x - \alpha_k)p_k(x) - \beta_k p_{k-1}(x)$$

then

$$\begin{pmatrix} \alpha_0 & 1 & 0 & 0 & \cdots & 0 \\ \beta_1 & \alpha_1 & 1 & 0 & \ddots & 0 \\ 0 & \beta_2 & \alpha_2 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \beta_{n-1} & \alpha_{n-1} & 1 \\ 0 & 0 & 0 & 0 & \beta_n & \alpha_n \end{pmatrix} \begin{pmatrix} p_0(x_l) \\ p_1(x_l) \\ p_2(x_l) \\ \vdots \\ p_{n-1}(x_l) \\ p_n(x_l) \end{pmatrix} = x_l \begin{pmatrix} p_0(x_l) \\ p_1(x_l) \\ p_2(x_l) \\ \vdots \\ p_{n-1}(x_l) \\ p_n(x_l) \end{pmatrix} \tag{3.70}$$

which means the matrix A on left-hand-side has $(n+1)$ eigen-pairs $\{x_l, (p_j(x_l))_{j=0}^n\}$. The tridiagonal matrix has a similar transform to a symmetric matrix, denoted by $S = D^{-1}AD$ with D as a diagonal matrix, then S has the same set of eigen-pairs, which means the transformed vectors

$$D^{-1} \begin{pmatrix} p_0(x_l) \\ p_1(x_l) \\ p_2(x_l) \\ \vdots \\ p_{n-1}(x_l) \\ p_n(x_l) \end{pmatrix} \tag{3.71}$$

are the eigenvectors of S. Since all the eigenvalues are distinct, these vectors are orthogonal, which implies

$$v_k^T D^{-2} v_j = r_j \delta_{jk} \quad r_j > 0. \tag{3.72}$$

where $v_j = (p_i(x_j))_{i=0}^n$. Take $f = p_k$ into the quadrature rule,

$$\sum_{j=0}^{n} c_j v_{jk} = \delta_{0k} \tag{3.73}$$

Combine the two equations, we have

$$\begin{aligned} \sum_{k=0}^{n} \sum_{j=0}^{n} c_j v_{jk} v_{lk} D_{kk}^{-2} &= \sum_{k=0}^{n} v_{lk} D_{kk}^{-2} \sum_{j=0}^{n} c_j v_{jk} = D_{00}^{-2} > 0 \\ &= \sum_{j=0}^{n} c_j \sum_{k=0}^{n} v_{lk} D_{kk}^{-2} v_{jk} = r_l c_l. \end{aligned} \tag{3.74}$$

□

This result should be compared with Newton-Cotes formula, where the weights are not all positive if n is large, hence Gauss quadrature has better numerical stability. At last, we briefly state the error estimate for the Gauss quadrature using the previous proved theorem 3.3.4, where $r = 2n + 1$.

Corollary 3.4.6. *For $f \in C^{2n+2}[-1, 1]$, the error of Gauss quadrature satisfies*

$$|\mathcal{I}(f) - \mathcal{I}_n(f)| \le \Omega_{2n+1} \frac{(b-a)^{2n+3}}{(2n+2)!} \max_{x\in[-1,1]} |f^{(2n+2)}(x)|. \tag{3.75}$$

Remark 3.4.7. *For Chebyshev polynomials. the weight $w(x) = (1 - x^2)^{-1/2}$, the Gauss quadrature nodes are roots of $T_{n+1}(x) = \cos((n + 1) \arccos x)$,*

$$x_j = \cos\left(\frac{2j+1}{2(n+1)}\pi\right), \quad c_j = \frac{\pi}{n+1}.$$

which is exactly the set of Chebyshev interpolation nodes. For Gauss-Lobatto nodes, they are

$$\tilde{x}_j = \cos\left(\frac{j\pi}{n}\right), \quad c_j = \frac{\pi}{d_j n}$$

where $d_j = 2$ if $j = 0$ or $j = n$, otherwise $d_j = 1$. This is exactly the composite trapezoid rule.

3.5 Gauss Quadrature on Unbounded Domain

For the integral over infinite interval $[0, \infty)$ or $(-\infty, \infty)$ using the Gauss quadrature, the weight needs to be decaying faster than polynomial growth. Usual choices are $w(x) = e^{-x}$ and $w(x) = e^{-x^2}$. The corresponding orthogonal polynomials are called Laguerre polynomials and Hermite polynomials (not interpolating polynomial), respectively.

Laguerre polynomial

The domain is $[0, \infty)$ and the Laguerre polynomials are defined by

$$p_n(x) = e^x \frac{d^n}{dx^n}(e^{-x}x^n), \quad n \ge 0 \tag{3.76}$$

It can be shown easily using integration by parts that $\langle p_n, p_m\rangle_w = 0$ if $n \ne m$. The polynomials satisfy the recursive formula

$$p_{n+1}(x) = (2n + 1 - x)p_n(x) - n^2 p_{n-1}(x) \tag{3.77}$$

The initial values are $p_{-1} = 0$ and $p_0 = 1$ as usual.

Hermite polynomial

The domain is $(-\infty, \infty)$ and the Hermite polynomials are defined by

$$h_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}. \tag{3.78}$$

The recursive formulation is

$$h_{n+1}(x) = 2xh_n(x) - 2nh_{n-1}(x).$$

3.6 Probabilistic Integration

3.6.1 Quasi Monte-Carlo Integration

3.7 Exercises

3.7.1 Theoretical Part

Problem 3.7.1. *Based on Theorem 3.2.4 and Theorem 3.2.5, derive the backward difference formula.*

Problem 3.7.2. *Estimate the error (with rounding error) for the following central difference scheme*

$$f'(x) \simeq \frac{-f(x+h) + 8f(x+h/2) - 8f(x-h/2) + f(x-h)}{6h}.$$

When $f(x) = \exp(x)$ on $[0,1]$, what value would be a suitable choice for h?

Problem 3.7.3. *Let the integration $\mathcal{I}(f)$ given by*

$$\mathcal{I}(f) = \int_0^1 x^\alpha f(x)dx, \quad \alpha \in [0,1]$$

and consider the quadrature formula $\mathcal{I}_0(f) = w_0 f(x_0)$. Is it possible to find an α that the quadrature rule has degree of accuracy of $r = 2$?

Problem 3.7.4. *Let $n \geq 2$ is even and $\{x_j\}_{j=0}^n$ the nodes for closed form Newton-Cotes formula on $[a,b]$. Define $\omega(x) = \prod_{j=0}^n (x - x_j)$ and $F(x)$ defined as*

$$F(x) = \int_a^x \omega(t)dt.$$

Prove that $F(a) = F(b) = 0$ and $F(x) > 0$ over (a,b). Hint: Notice F is symmetric. First show that $|\omega(x)| > |\omega(x+h)|$ when $x \in (a, a+h)$, where $h = (b-a)/n$.

Problem 3.7.5. *Let $f(x) \in C(\mathbb{R})$ be a 2π-periodic function and f_n is the interpolating trigonometric polynomial on equally-spaced nodes $x_j = \frac{2\pi j}{2n+1}$, $j = 0, \ldots, 2n$. Show that*

$$\mathcal{I}(f_n) = \sum_{j=0}^{2n} w_j f(x_j). \tag{3.79}$$

for positive weights $w_j > 0$, $j = 0, \ldots, 2n$. Hint: show that the "Lagrange basis" are $l_j(x) = \frac{1}{2n+1}\frac{\sin((2n+1)(x-x_j)/2)}{\sin((x-x_j)/2)}$.

3.7.2 Computational Part

Problem 3.7.6. *Let the infinite matrix A such that*

$$A_{jk} = \frac{1}{(j+k-1)(j+k)/2 - (k-1)}.$$

Compute the operator norm $\|A\|$ with 10 digits accuracy. Hint: use extrapolation.

Problem 3.7.7. *Implement the $\underline{\text{adaptive}}$ integration with composite Newton-Cotes closed formula for $n = 4$.*

Chapter 4

Approximation

The approximation solves the problem

$$\min_{p\in P} \|f-p\|$$

which is to select the function $p \in P$ in a specific set that has the minimum distance from the target function f.

4.1 General Approximation Theory

The most famous example in approximation theory is the Least Square problem

$$\min_{x\in S} \|Ax-b\|_2$$

where $A \in \mathbb{R}^{N\times k}$ and a given vector $b \in \mathbb{R}^N$. Seeking for solution $x \in S = \mathbb{R}^k$ is the simplest case, in general the problem can be efficiently solved if S is a convex set.

Definition 4.1.1. *Let $\mathcal{M} \subset \mathcal{V}$ of a normed space $(\mathcal{V}, \|\cdot\|)$ and given $v \in \mathcal{V}$, the best approximation in $\|\cdot\|$ is*

$$u^* \in \mathcal{M}, \quad \|u^*-v\| = \inf_{u\in\mathcal{M}} \|u-v\|$$

Definition 4.1.2. *The sequence u_k, $k \in \mathbb{N}$ is an minimizing sequence if*

$$u_k \in \mathcal{M}, \quad \|u_k - v\| \to \inf_{u\in\mathcal{M}} \|u-v\|, \quad k \to \infty. \tag{4.1}$$

Theorem 4.1.3 (existence of best approximation). *If u_k is a minimizing sequence and has an accumulation point u^* in $\mathcal{M}$, then u^* is a best approximation to v.*

Proof. Just take limit (subsequence) on both sides to

$$\|u^*-v\| \le \|u^*-u_k\| + \|u_k - v\| \tag{4.2}$$

□

Theorem 4.1.4. *If $\mathcal{M}$ is a compact subset of $\mathcal{V}$, then best approximation always exists.*

One special case is that $\mathcal{M}$ is a finite dimension linear subspace of $\mathcal{V}$, then one can take a bounded closed set truncating the minimizing sequence, then such set must be compact.

Lemma 4.1.5 (convexity). *If $\mathcal{M}$ is a convex set of normed space $\mathcal{V}$, then the set of best approximations is convex.*

Theorem 4.1.6 (uniqueness). *If $\mathcal{M}$ is strictly convex of normed space $\mathcal{V}$, then the best approximation is unique. It is worthwhile to notice the strictly convexity is sufficient but not necessary.*

Definition 4.1.7. *The normed space $(\mathcal{V}, \|\cdot\|)$ is strictly normed if and only if unit ball is strictly convex).*

Theorem 4.1.8 (uniqueness). *If $\mathcal{M}$ is a strictly normed linear subspace of normed space $\mathcal{V}$, then there exists at most one best approximation for each $v \in \mathcal{V}$.*

4.2 Minimax Approximation

Given $f \in C^0([a,b])$, find a polynomial $p_n \in \Pi_n$ such that

$$\|f - p_n\|_\infty = \min_{g\in\Pi_n} \|f - g\|_\infty$$

Such problem is the minimax approximation problem. In the previous sections, we have seen a similar problem which is to minimize the maximum of $|\omega(x)|$ with $\omega(x) = \prod_{j=0}^n (x - x_j)$. The proof used there can be borrowed for the following theorem as well.

Theorem 4.2.1 (de la Vallée-Poussin). *Let $f \in C^0([a,b])$ and $n \geq 0$, let $x_0 < x_1 < \cdots < x_{n+1}$ be $n+2$ nodes in $[a,b]$. If there exists a polynomial $q_n \in \Pi_n$ such that*

$$f(x_j) - q_n(x_j) = (-1)^j e_j, \quad j = 0, \ldots, n+1.$$

where e_j are having the same sign and nonzero, then

$$\min_j |e_j| \leq E_n^*(f) := \min_{g\in\Pi_n} \|f - g\|_\infty \leq \max_j |e_j|. \tag{4.3}$$

Proof. Prove by contradiction. Suppose $E_n^*(f)$ is achieved by some polynomial $g \in \Pi_n$. If q_n satisfies that

$$|e_j| > E_n^*(f) = \|f - g\|_\infty. \tag{4.4}$$

Then $q_n - g = q_n - f - (g - f)$, which implies that

$$\operatorname{sgn}(q_n - g)(x_j) = \operatorname{sgn}(q_n - f)(x_j) = -(-1)^j \operatorname{sgn}(e_j) \tag{4.5}$$

which changes sign $n+1$ times while $q_n - g$ only has n roots. □

Theorem 4.2.2 (Chebyshev equioscillation theorem). *Let $f \in C^0([a,b])$ and $n \geq 0$. Then there exists a unique polynomial $q^* \in \Pi_n$ such that*

$$E_n^*(f) = \|f - q_n^*\|_\infty \tag{4.6}$$

This polynomial is uniquely characterized by the property:

$$a \leq x_0 < \cdots < x_{n+1} \leq b$$

for which we can select $\sigma = \pm 1$ that

$$f(x_j) - q_n^*(x_j) = \sigma(-1)^j E_n^*(f), \quad j = 0, 1, \dots, n+1. \tag{4.7}$$

Proof. The existence of such minimizing polynomial $q^* \in \Pi_n$ can be proved through a minimizing sequence argument. Let $q^k \in \Pi_n$ be a minimizing sequence to having $\|q^k - f\| \to E_n^*(f)$, then it is clear that the set

$$\mathcal{M} = \{q \in \Pi_n \mid \|q - f\| \leq E_n^*(f) + 1\} \tag{4.8}$$

must be non-empty and also such set is compact since $\mathcal{M}$ is of finite dimension and closed. Then the minimizing sequence will have a converging sub-sequence such that the limit sits in $\mathcal{M}$.

Then we show the alternation property for this minimizing polynomial. If there is no alternation between f and q_n, then we can partition the interval $[a,b]$ into $1 \leq N \leq n+1$ parts

$$[a,b] = \cup_{j=1}^N [t_{j-1}, t_j], \quad a = t_0 < \cdots < t_N = b.$$

that

1. $f(t_k) - q_n(t_k) = 0$, $k = 1, \dots, N-1$.
2. for each $1 \leq k \leq N$, there exists $s_k \in [t_{k-1}, t_k]$ such that

   $$|f(s_k) - q_n(s_k)| = \|f - q_n\|_\infty \neq 0$$

   and for any $x \in [t_{k-1}, t_k]$,

   $$-(f(x) - q_n(x)) \neq (f(s_k) - q_n(s_k)).$$

3. for each $1 \leq k \leq N-1$,

   $$f(s_k) - q_n(s_k) = -(f(s_{k+1}) - q_n(s_k),$$

Without loss of generality, one can assume that

$$\text{sgn}(f(s_k) - q_n(s_k)) = (-1)^{k-1} \tag{4.9}$$

then using the second condition, there exists ε such that $\forall x \in [t_{k-1}, t_k]$,

$$\begin{aligned} -\|f - q_n\|_\infty + \varepsilon \leq f(x) - q_n(x) \qquad & k \text{ is odd} \\ f(x) - q_n(x) \leq \|f - q_n\|_\infty - \varepsilon \qquad & k \text{ is even} \end{aligned} \tag{4.10}$$

Then we construct a polynomial $g \in \Pi_{N-1}$ that

$$\mathrm{sgn}(g(x)) = (-1)^k, \quad x \in [t_{k-1}, t_k]. \tag{4.11}$$

and $\|g\| \le \frac{\varepsilon}{2}$ and let $k(x) = q_n(x) - g(x) \in \Pi_n$ and $f(x) - k(x) = f(x) - q_n(x) + g(x)$, which implies that

1. $f(x) - k(x) < f(x) - q_n(x)$, if $x \in (t_{k-1}, t_k)$ for k odd.
2. $f(x) - k(x) \ge -\|f - q_n\| + \frac{\varepsilon}{2}$, if $x \in (t_{k-1}, t_k)$ for k odd.
3. $f(x) - k(x) > f(x) - q_n(x)$, if $x \in (t_{k-1}, t_k)$ for k even.
4. $f(x) - k(x) \le \|f - q_n\| - \frac{\varepsilon}{2}$, if $x \in (t_{k-1}, t_k)$ for k odd.

In this way, $\|f - k\|_\infty < \|f - q_n\|_\infty$, contradiction.

In the last, we show the uniqueness. Suppose there are two polynomials $q_n, \tilde{q}_n$ being the minimax approximation. Then $\frac{1}{2}(q_n + \tilde{q}_n)$ must also be a minimax approximation, therefore there exists the alternation nodes

$$a \le s_0 < s_1 < \cdots < s_{n+1} \le b \tag{4.12}$$

such that

$$\frac{1}{2}(q_n - f)(s_k) + \frac{1}{2}(\tilde{q}_n - f)(s_k) = \sigma(-1)^k E_n^*(f) \tag{4.13}$$

Therefore, we must have

$$(q_n - f)(s_k) = (\tilde{q}_n - f)(s_k) \Rightarrow q_n(s_k) = \tilde{q}_n(s_k). \tag{4.14}$$

□

4.2.1 Remez Algorithm

The numerical method to find the minimax polynomial is using the idea of the above Chebyshev equioscillation theorem. Intuitively, the aiming polynomial will be oscillatory comparing with f, therefore we can start with the Chebyshev polynomial approximation, which is

$$C_n(x) = \sum_{j=0}^{n} c_j T_j(x), \quad c_j = \langle f, T_j \rangle_w / \langle T_j, T_j \rangle_w \tag{4.15}$$

where $w = (1 - x^2)^{-1/2}$. The convergence is uniform as $n \to \infty$, especially if $f \in C^r([a, b])$, the coefficient $c_j \sim \mathcal{O}(j^{-r})$ decays fast, then the reminder approximately $f - C_n \simeq c_{n+1}T_{n+1}$, this reminder achieves equioscillation at exactly $(n+2)$ points. The Remez algorithm is based on the above idea and perform an iterative construction.

1. Initialize local extrema nodes x_j of T_{n+1}.

2. Solve the equation that
$$\sum_{j=0}^{n} a_j x_k^j + (-1)^k E = f(x_k) \tag{4.16}$$
for unknown coefficients a_j and the estimated error E, $k = 0, \ldots, n+1$.

3. Form a new polynomial $p(x) = \sum_{j=0}^{n} a_j x^j$.

4. Locate the local extrema nodes of $|p(x) - f(x)|$ as a new set of nodes (if no local extrema on the sub-interval, then take the larger magnitude on end nodes). If equioscillation condition is nearly achieved, then stop. Otherwise repeat step 2 with the new extrema nodes.

4.3 Orthogonal Polynomials

The orthogonal polynomials are actually a special case of generalized Fourier series. Let the weight function $w(x) \geq 0$ on the interval $(-1, 1)$ be an integrable function. Then we can define a sequence of polynomials p_k, deg$(p_k) = k$ and
$$\int_{-1}^{1} w(x) p_k(x) p_j(x) dx = 0, \quad \text{if } k \neq j. \tag{4.17}$$

Here we define the inner product $\langle f, g \rangle_w$ by
$$\langle f, g \rangle_w = \int_{-1}^{1} w(x) f(x) g(x) dx \tag{4.18}$$

This inner product induces a norm $\|f\|_w = \sqrt{\langle f, f \rangle_w}$, we then define the corresponding space as
$$L_w^2 = \{f : (-1, 1) \to \mathbb{R} \mid \|f\|_w < \infty\}. \tag{4.19}$$

Then for any $f \in L_w^2$, we can define the generalized Fourier series by Sf:
$$Sf = \sum_{j=0}^{\infty} a_j p_j, \quad a_j = \frac{\langle f, p_j \rangle_w}{\langle p_j, p_j \rangle_w}. \tag{4.20}$$

The series converges to f in L_w^2 sense from the Parseval's equality:
$$\|f\|_w^2 = \sum_{j=0}^{\infty} a_j^2 \|p_j\|_w^2. \tag{4.21}$$

The truncated series f_n:
$$f_n = \sum_{j=0}^{n} a_j p_j$$

is the best degree-n polynomial approximation to f, that is

$$\|f - f_n\|_w = \min_{q \in \Pi_n} \|f - q\|_w.$$

The polynomial f_n is the orthogonal projection of f onto Π_n in the sense of L^2_w.

The generation of p_j follows from the following recursive formula

$$p_{j+1} = (x - \alpha_j)p_j(x) - \beta_j p_{j-1}(x), \quad j \geq 0 \tag{4.22}$$

The initial conditions are $p_{-1} = 0, p_0 = 1$. The constants α_j and β_j can be obtained by noticing that

$$\begin{aligned} \langle p_{j+1}, p_j \rangle_w = 0 \Rightarrow \alpha_j &= \frac{\langle x p_j, p_j \rangle_w}{\langle p_j, p_j \rangle_w} \\ \langle p_{j+1}, p_{j-1} \rangle_w = 0 \Rightarrow \beta_j &= \frac{\langle p_j, p_j \rangle_w}{\langle p_{j-1}, p_{j-1} \rangle_w} \end{aligned} \tag{4.23}$$

Here p_j are not normalized.

4.3.1 Chebyshev Polynomials

The Chebyshev polynomials are generated by using the weight $w(x) = (1-x^2)^{-1/2}$ on $(-1,1)$. The corresponding space is

$$L^2_w = \{f : (-1,1) \to \mathbb{R} \mid \int_{-1}^{1} f^2(x)(1-x^2)^{-1/2} dx < \infty\}$$

It is clear that by setting $p_k(x) = \cos(k \arccos x)$, the integral

$$\int_{-1}^{1} p_k(x)p_j(x)(1-x^2)^{-1/2} dx = \int_0^{\pi} \cos(k\theta)\cos(j\theta) d\theta = \begin{cases} \pi & k = j = 0 \\ \frac{\pi}{2} & k = j \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

Therefore the generalized Fourier series with Chebyshev polynomials is

$$f(x) = \sum_{j=0}^{\infty} a_j p_j(x)$$

where $p_j(x) = T_j(x)$ the j-th Chebyshev polynomial and $a_j = \frac{1}{\pi}\langle f, p_j \rangle_w$ if $j = 0$ and $a_j = \frac{2}{\pi}\langle f, p_j \rangle_w$ otherwise.

4.3.2 Legendre Polynomial

The Legendre polynomials are generated by using the weight $w(x) = 1$. The corresponding space is the normal $L^2(-1,1)$. The recursive formula for Legendre polynomials is

$$L_{j+1}(x) = \frac{2j+1}{j+1} x L_j(x) - \frac{j}{j+1} L_{j-1}(x) \tag{4.25}$$

and $\langle L_j, L_j\rangle_w = \frac{2}{2j+1}$. Therefore the generalized Fourier series is

$$f(x) = \sum_{j=0}^{\infty} a_j L_j(x), \quad a_j = \frac{2j+1}{2}\langle f, L_j\rangle_w. \tag{4.26}$$

The first a few Legendre polynomials are

$$L_0 = 1, \quad L_1(x) = x, \quad L_2(x) = \frac{1}{2}(3x^2 - 1). \tag{4.27}$$

Remark 4.3.1. *Both of above examples are special cases of Jacobi polynomials which are generated by weight function* $w(x) = (1-x)^\alpha(1+x)^\beta$.

4.4 Approximation Theory for Compact Groups

4.5 Padé Approximation

4.6 Neural Network

4.6.1 Radial Basis

4.6.2 Universal Approximation Theorem

4.6.3 Gradient-Flow

4.7 Exercises

4.7.1 Theoretical Part

Problem 4.7.1. *Show that the vector space* $C^0([a,b])$ *with* $\|\cdot\|_\infty$ *is not strictly normed.*

Problem 4.7.2 (interlacing). *Let* p_n *be the orthogonal polynomials with respect to weight function* $w(x) \geq 0$, *prove the zeros of* p_n *and* p_{n+1} *are alternating.*

$$-1 < x_{1,n+1} < x_{1,n} < x_{2,n+1} < x_{2,n} < \cdots < x_{n,n+1} < x_{n,n} < x_{n+1,n+1} < 1. \tag{4.28}$$

where $x_{j,k}, 1 \leq j \leq k$ *are the zeros of* p_k. *Hint: use the recursive formula.*

Problem 4.7.3. *Let* $f(x) \in C^0([-1,1])$ *is even (odd). Show that the minimax approximation* $q_n(x) \in \Pi_n$ *is also even (odd). Hint: think about the function* $\frac{1}{2}(q_n(x) \pm q_n(-x))$, *then use Chebyshev equioscillation theorem and the uniqueness.*

4.7.2 Computational Part

Problem 4.7.4. *Implement Remez algorithm and find the nearly minimax approximation in* Π_3 *for the function* $f(x) = e^x$ *on* $[0,1]$.

Problem 4.7.5. *Implement the Gauss quadrature* $n = 4$ *and compare the accuracy with Newton-Cotes closed formula* $n = 4$ *for* $f(x) = e^x$ *on* $[0,1]$.

Chapter 5

Ordinary Differential Equations

The ordinary differential equations in the form

$$y' = f(y,t), \quad y(a) = y_0$$

are commonly used in modeling for various fields of physical sciences and biological studies. In this chapter we will derive and analyze the numerical methods for solving the above problems for ordinary differential equations.

5.1 Initial Value Problem

Let $f(y,t) : \mathbb{R}^n \times [a,b] \to \mathbb{R}^N$ be a continuous function, the initial value problem is to determine a differentiable solution y satisfying that

$$y' = f(y,t), \quad y(a) = y_0 \tag{5.1}$$

the derivative $y' \in \mathbb{R}$ denotes taking derivatives on all components of y. The general existence and uniqueness of solution to ODE (5.1) depends on the Lipschitz condition of f

$$\|f(y_1,t) - f(y_2,t)\| \le L\|y_1 - y_2\|, \quad t \in [a,b].$$

Theorem 5.1.1. *If f is L-Lipschitz continuous, then*

1. *The ODE (5.1) has a unique continuously differentiable solution $y : [t_0, t_1] \mapsto \mathbb{R}^N$.*
2. *The stability with respect to initial condition is Lipschitz,*

$$\|y(t) - \tilde{y}(t)\| \le e^{L(t-t_0)}\|y(t_0) - \tilde{y}(t_0)\|$$

where $y(t_0), \tilde{y}(t_0)$ are the initial conditions for y and $\tilde{y}$.

Remark 5.1.2. *Consider a perturbative ODE:*

$$y' = f(y,t) + \delta(t), \quad y(a) = y_0 + \delta_0 \tag{5.2}$$

such that $\|\delta(t)\|, \|\delta_0\| \le \varepsilon$ for all $t \in [a,b]$ and the resulting perturbative equation's solution $\tilde{y}$ satisfies

$$\|y(t) - \tilde{y}(t)\| \le C\varepsilon$$

then the problem is Liapunov stable. The Lipschitz continuity of f guarantees the Liapunov stability.

5.1.1 One-step Methods

The simplest numerical solution for IVP is the one-step method. We consider the discretization of solution

$$y_n \simeq y(t_n), \quad n = 0, 1, \ldots, M$$

on the grid

$$\Delta = \{a = t_0 < t_1 < \cdots < t_M = b\}, \quad h_n := t_{n+1} - t_n.$$

Definition 5.1.3 (scheme). *The explicit (forward) one-step method for approximation of the solution to the IVP (5.1) is in the from*

$$y_{n+1} = y_n + h_n \phi(t_n, y_n; h_n), \quad y_0 := y(t_0). \tag{5.3}$$

where ϕ is the iteration function.

Definition 5.1.4 (convergence). *The convergence order is p if the global error*

$$\max_{n \geq 0} \|y_n - y(t_n)\| \leq C \bar{h}^p, \quad \bar{h} := \max_{n \geq 0} h_n \tag{5.4}$$

where C is independent of the grid Δ.

Definition 5.1.5 (local error). *The local error $E(t,h)$ (the quantity $E(t,h)/h$ is the local truncation error)*

$$E(t,h) := y(t) + h\phi(t, y(t); h) - y(t+h), \quad 0 \leq h \leq b - t \tag{5.5}$$

denotes the induced error by one-step method at time $t + h$.

Definition 5.1.6 (consistency). *The consistency order is p if*

$$\|E(t,h)\| \leq C h^{p+1}, \quad t \in [a,b], \quad 0 \leq h \leq b - t, \tag{5.6}$$

where C is independent of t, h.

Example 5.1.7 (Forward Euler). *If we let $\phi(t, y; h) = f(t, y) \in C^1([a,b] \times \mathbb{R}^N; \mathbb{R}^N)$, then we will have the local error in the form*

$$\begin{aligned} E(t,h) &= y(t) + hf(t, y(t)) - y(t+h) \\ &= y(t) + hf(t, y(t)) - (y(t) + hy'(t) + \frac{h^2}{2} y''(\xi)) \\ &= -\frac{h^2}{2} y''(\xi). \end{aligned} \tag{5.7}$$

which means consistency order is $p = 1$.

Example 5.1.8 (modified Euler). *Let the iteration function*

$$\phi(t, y; h) = a_1 f(t, y) + a_2 f(t + b_1 h, y + b_2 h f(t, y))$$

then local error

$$\begin{aligned}E(t,h) &= y(t) + a_1 h f(t,y(t)) + a_2 h \left[f(t,y(t)) + \partial_t f|_{t,y(t)} b_1 h + \partial_y f|_{t,y(t)} b_2 h + \mathcal{O}(h^2)\right] \\ &\quad - (y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \frac{h^3}{6}y'''(\xi)) \\ &= (a_1 + a_2 - 1) f(t,y(t)) + h^2\left(a_2 b_1 \partial_t f|_{t,y(t)} + a_2 b_2 \partial_y f|_{t,y(t)} - \frac{1}{2}y''(t)\right) + \mathcal{O}(h^3)\end{aligned}$$

Therefore if the constants

$$a_1 + a_2 = 1, \quad a_2 b_1 = \frac{1}{2}, \quad a_2 b_2 = \frac{1}{2},$$

the consistency order is $p = 2$. This formulation cannot obtain a formula for $p = 3$.

1. *The modified Euler is $a_1 = 0$, $a_2 = 1$, $b_1 = b_2 = \frac{1}{2}$. The algorithm can be divided into two updating sub-steps:*

$$\begin{aligned}y_{n+1/2} &= y_n + \frac{1}{2}h_n f(t_n, y_n), \quad t_{n+1/2} = t_n + \frac{1}{2}h_n, \\ y_{n+1} &= y_{n+1/2} + \frac{1}{2}h_n f(t_{n+1/2}, y_{n+1/2})\end{aligned} \tag{5.8}$$

2. *The Heun's method is $a_1 = \frac{1}{2}$, $a_2 = \frac{1}{2}$, $b_1 = b_2 = 1$. This method can be written into similar sub-steps as well.*

Remark 5.1.9. *For implicit methods, the iteration function involves unknown values. Two famous examples are implicit Euler:*

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1})$$

and Crank-Nicolson:

$$y_{n+1} = y_n + \frac{h_n}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1})).$$

Theorem 5.1.10. *If the one-step method has consistency order $p \geq 1$ and $\phi(t,y;h)$ is Lipschitz continuous in y variable, then convergence order is p.*

Proof. Let $e_n = y_n - y(t_n)$, then using the local error relation that

$$y(t_{n+1}) = y(t_n) + h_n \phi(t_n, y(t_n); h_n) - E(t_n, h_n) \tag{5.9}$$

we have

$$e_{n+1} = e_n + h_n(\phi(t_n, y_n; h_n) - \phi(t_n, y(t_n); h_n)) + E(t_n, h_n) \tag{5.10}$$

which implies that

$$\begin{aligned}\|e_{n+1}\| &\leq \|e_n\| + h_n L \|y_n - y(t_n)\| + C h_n^{p+1} \\ &\leq (1 + h_n L)\|e_n\| + C h_n^{p+1} \\ &\leq e^{h_n L}\|e_n\| + C h_n^{p+1} \\ &\leq e^{h_n L} e^{h_{n-1} L}\|e_{n-1}\| + e^{h_{n-1}L} C h_n^{p+1} + C h_{n-1}^{p+1} \\ &\dots \\ &\leq e^{(b-a)L}\|e_0\| + C e^{(b-a)L}\sum_{n\geq 0} h_n^{p+1} \\ &\leq C(b-a)e^{L(b-a)}\bar{h}^p.\end{aligned} \tag{5.11}$$

□

Example 5.1.11 (explicit Runge Kutta). *The Runge Kutta method RK4 has convergence order $p = 4$ uses ϕ as*

$$\phi(t, y; h) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k4) \tag{5.12}$$

where

$$\begin{aligned} k_1 &= f(t, y), \\ k_2 &= f(t + \frac{h}{2}, y + \frac{h}{2}k_1), \\ k_3 &= f(t + \frac{h}{2}, y + \frac{h}{2}k_2), \\ k_4 &= f(t + h, y + hk_3). \end{aligned} \tag{5.13}$$

The proof of consistency order uses Taylor's expansion, see exercises.

5.1.2 Absolute Stability

In the previous section, we have discussed the so-called zero-stability, which shows that small perturbation (comparable to step size) during each step will not cause too much trouble. This stability is with respect to perturbation. The absolute stability is to keep the numerical solution bounded (actually decaying) as $n \to \infty$, which is an asymptotic behavior. The usual toy problem is

$$y'(t) = f(t, y) := \lambda y(t), \quad t > 0 \tag{5.14}$$

where $\lambda \in \mathbb{C}$ and initial condition is $y(0) = 1$. The solution is simple $y(t) = e^{\lambda t}$, of course if $\Re\lambda > 0$, the solution would blow and the solution will decay to zero if $\Re\lambda < 0$.

Definition 5.1.12. *A numerical scheme is absolutely stable if*

$$y_n \to 0, \quad n \to \infty,$$

where the time step is fixed and y_n is the numerical solution to (5.14) under the given scheme. The region of absolute stability refers to the value of $z = h\lambda$ such that

$$\mathcal{A} = \{z = h\lambda \in \mathbb{C} \mid y_n \to 0, n \to \infty\} \tag{5.15}$$

Example 5.1.13 (forward Euler). *With the scheme that*

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n$$

we can easily find $y_n = (1 + h\lambda)^n = (1 + z)^n$. This value decays to zero as long as

$$|1 + z| < 1.$$

This region is a unit disk centered at $z = -1$, which means

$$h\lambda \in \mathbb{C}^-, \quad h \in (0, -2\frac{\Re\lambda}{|\lambda|^2}).$$

This shows that only when the step size is small enough, we can make sure the solution reflects the correct behavior of the solution's exponential decay.

Example 5.1.14. *Similarly, consider the implicit Euler, we will have*

$$y_n = (1 - h\lambda)^{-n}$$

then $|1 - z| < 1$ *derives* $\mathcal{A} \supset \mathbb{C}^-$. *This means when* $\lambda < 0$, *the numerical solution's behavior will be decaying as well.*

The numerical scheme is called A-stable if $\mathcal{A} \supset \mathbb{C}^-$. It can be shown that no explicit linear schemes can be A-stable **?**.

5.1.3 Rounding Error

In this section, we briefly discuss the impact of rounding error. Following the previous theorem about the local error, if each step involves rounding error of size $|\delta_n| \le \delta$

$$y_{n+1} = y_n + h_n\phi(t_n, y_n; h_n) + \delta_n \tag{5.16}$$

then e_{n+1} satisfies

$$\begin{aligned} \|e_{n+1}\| &= \|e_n + h_n(\phi(t_n, y_n; h_n) - \phi(t_n, y(t_n); h_n)) + \delta_n + E(t_n, h_n)\| \\ &\le e^{(b-a)L}\|e_0\| + Ce^{(b-a)L}\sum_{n\ge 0}(h_n^{p+1} + \delta) \\ &\le e^{(b-a)L}\|\delta_0\| + Ce^{(b-a)L}(b-a)\left(\bar{h}^p + \frac{\delta}{\underline{h}}\right) \end{aligned} \tag{5.17}$$

where $\underline{h} = \min_{n\ge 0} h_n$. When the grid is uniform that $h_n = h$, then the optimal grid size is minimizing $h^p + \frac{\delta}{h}$.

5.1.4 Extrapolation Methods

When the grid Δ is uniform, it is known that

$$y(t_n) - y_n = \mathcal{O}(h^p),$$

where p is the consistency order. Actually, one can show that $y(t_n) - y_n$ can be written in an asymptotic expansion. For a proof of the following theorem, we refer to **?**.

Theorem 5.1.15. *Let f and ϕ be $(p + r)$-times continuously partial differentiable in each variable. Then there exists coefficients $c_{p+j} \in C^{r+1-j}([a, b], \mathbb{R}^N)$ that $c_{p+j}(a) = 0$ for $j = 0, 1, \dots, r - 1$ and*

$$y(t_n) - y_n = \sum_{j=0}^{r-1} c_{p+j}(t_n)h^{p+j} + \mathcal{O}(h^{p+r}). \tag{5.18}$$

In order to find solution's value $y(t_n)$ with higher order of error, one can simply apply Richardson extrapolation.

5.1.5 Adaptive One-step Methods

The idea of adaptive one-step method is similar to the adaptive quadrature. The basic algorithm is briefly outlined in the following:

1. Using an initial uniform grid Δ with grid size of h, one can compute the numerical solution y_n with error of $e_h = \mathcal{O}(h^p)$.
2. For each time step from t_{n-1} to t_n, estimate the error of $e_h = y(t_n) - y_n$ from extrapolation, which is approximated by
$$y_n(h) - z_n(h) \simeq (1 - 2^{-p})e_h$$
where $z_n(h)$ is computed through a refined one-step method locally,
$$\begin{aligned} w_n(h) &= y_n(h) + \frac{h}{2}\phi(t_n, y_n; h/2), \\ z_n(h) &= w_n(h) + \frac{h}{2}\phi(t_n + \frac{h}{2}, w(h); \frac{h}{2}). \end{aligned} \tag{5.19}$$
3. If the error is within certain tolerance threshold, then continue to next time step, otherwise sub-divide $[t_{n-1}, t_n]$ and perform the adaptive one-step method.

5.1.6 Multistep Methods

An m-step method for the ODE (5.1) on uniform grid is formulated by

$$\sum_{j=0}^{m} c_j y_{l+j} = h\phi(t_l, y_l, \ldots, y_{l+m}; h) \tag{5.20}$$

The coefficients $c_j \in \mathbb{R}$ and the leading coefficient $c_m \neq 0$. Clearly the one-step method is a special case that $c_0 = -1$ and $c_1 = 1$. The time stamps are equally spaced $t_l = a + lh$ for $l = 0, \ldots, n$, where $h = \frac{b-a}{n}$. The initial values $y_0, \ldots, y_{m-1}$ are assumed known.

Remark 5.1.16. *Usually, the initial value y_0 is given, while the other initial values are not. The preparation of the remaining values $y_1, \ldots, y_{m-1}$ can be done through the aforementioned one-step methods. The m-step method can also be implicit if unknown values are involved.*

Example 5.1.17. *The central difference explicit scheme*

$$y_{n+1} = y_{n-1} + 2hf(t_n, y_n)$$

Intuitively,

$$y_{n+1} = y_{n-k} + \int_{t_{n-k}}^{t_{n+1}} f(t, y(t))dt \tag{5.21}$$

and one can replace the integral with any quadrature rule involving nodes t_j in between (end nodes included, but could be implicit).

We can define the convergence and consistency for m-step methods similarly by extending one-step methods.

Definition 5.1.18 (convergence). *The convergence order is p if the scheme satisfies*

$$\|y(t_n) - y_n\| \le Ch^p$$

for an independent C for all n (including initial values).

Definition 5.1.19 (local error). *The local error is*

$$E(t,h) = \left(\sum_{j=0}^{m} c_j y(t+jh)\right) - h\phi(t, y(t), y(t+h), \ldots, y(t+mh); h) \tag{5.22}$$

$E(t,h)/h$ is local truncation error. Here we normalize the formula by setting $c_m = 1$.

Definition 5.1.20 (consistency). *The consistency order is p if local error*

$$\|E(t,h)\| \le Ch^{p+1}.$$

For the following context, we assume that *ϕ is Lipschitz for all variables except the first one*, clearly the linear multi-step methods satisfies this condition.

Definition 5.1.21 (root condition). *The generating polynomial with respect to multi-step method is*

$$p(\xi) := \sum_{j=0}^{m} c_j \xi^m \tag{5.23}$$

The Dahlquist's root condition is that the roots of $p(\xi)$ satisfies

$$p(\xi) = 0 \Rightarrow |\xi| \le 1 \tag{5.24}$$

if a root $|\xi| = 1$, then it is simple.

Theorem 5.1.22. *If root condition is satisfied, then*

$$\|y_l - y(t_l)\| \le C\left(\max_{0\le j\le m-1} \|y_j - y(t_j)\| + \max_{t\in[a.b-mh]} \|E(t,h)\|/h\right) \tag{5.25}$$

Proof. Similar to one-step method, we have

$$\begin{aligned} \sum_{j=0}^{m} c_j y(t_{l+j}) &= h\phi(t_l, y(t_l), \ldots, y_{t_{l+m}}; h) + E(t_l, h) \\ \sum_{j=0}^{m} c_j y_{l+j} &= h\phi(t_l, y_l, \ldots, y_{l+m}; h) \end{aligned} \tag{5.26}$$

Take the difference, then

$$\sum_{j=0}^{m} c_j e_{l+j} = \underbrace{h\left[\phi(t_l, y_l, \dots, y_{l+m}; h) - \phi(t_l, y(t_l), \dots, y_{t_{l+m}}; h)\right]}_{\delta_l} - E(t_l, h) \tag{5.27}$$

The right-hand-side can be reviewed as certain perturbation and left-hand-side is a linear recursive scheme. This can be made into matrix form

$$\underbrace{\begin{pmatrix} e_{l+1} \\ e_{l+2} \\ \vdots \\ e_{l+m} \end{pmatrix}}_{E_l} = \underbrace{\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ -c_0 & \cdots & \cdots & \cdots & -c_{m-1} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} e_l \\ e_{l+1} \\ \vdots \\ e_{l+m-1} \end{pmatrix}}_{E_l} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ \vdots \\ \delta_l - E(t_l, h) \end{pmatrix}}_{\Delta_l} \tag{5.28}$$

The iteration then can be written as

$$E_l = A^l E_0 + \sum_{j=0}^{l-1} A^{l-1-j} \Delta_j \tag{5.29}$$

If E_l does not below, we need all eigenvalues λ_k of A satisfy $|\lambda_k| \le 1$, which are the roots of the polynomial (use elementary operations)

$$\det(\lambda I - A) = 0$$

That is exactly the generating polynomial:

$$p(\lambda) = \sum_{j=0}^{m} c_j \lambda^m \tag{5.30}$$

When the root condition is satisfied, for the $|\lambda_k| < 1$, those components will decay exponentially. If $|\lambda_k| = 1$, its geometric multiplicity has to be equal to algebraic multiplicity (then the Jordan block is diagonal), otherwise one Jordan block will be with size greater than one,

$$J_k = \begin{pmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{pmatrix} \tag{5.31}$$

and

$$\begin{pmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{pmatrix}^s \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} s\lambda_k^{s-1} \\ \lambda_k^s \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{5.32}$$

which will lead to unbounded behavior of A^l. Of course the root condition eliminates this case. Therefore

$$\|E_l\|_\infty \le C\left(\|E_0\|_\infty + \sum_{j=0}^{l-1}\|\Delta_j\|_\infty\right) \tag{5.33}$$

where Δ_s is bounded by the Lipschitz condition

$$\begin{aligned}\|\Delta_s\|_\infty &\le \|E(t_s,h)\| + hL\sum_{j=0}^{m}|e_{s+j}| \\ &\le \|E(t_s,h)\| + hLm\|E_s\|_\infty + hL\|E_{s+1}\|_\infty\end{aligned} \tag{5.34}$$

Hence

$$\sum_{s=0}^{l-1}\|\Delta_s\|_\infty \le l\max_s\|E(t_s,h)\| + hL(m+1)\sum_{s=0}^{l-1}\|E_s\|_\infty + hL\|E_l\|_\infty \tag{5.35}$$

It is then easy to see when h is small, the error E_l is bounded by E_0 and $E(t_s,h)$ (note $l \le n \simeq \mathcal{O}(h^{-1})$). □

Corollary 5.1.23. *If root condition is satisfied, then consistency order p implies convergence order p for suitable step size h.*

For consistency order, the main tool is the Taylor expansion, we carry out the following lemma without providing a proof (see exercise).

Lemma 5.1.24. *The linear m-step method*

$$\sum_{j=0}^{m} c_j y_{l+j} = h\sum_{j=0}^{m}\beta_j f(t_{l+j}, y_{l+j}) \tag{5.36}$$

has consistency order p if

$$\sum_{j=0}^{m}(j^s c_j - s j^{s-1}\beta_j) = 0, \quad s = 0,1,\ldots,p. \tag{5.37}$$

Proof. The proof is simple. Here we need f to be p-times continuously differentiable. □

5.1.7 Adams Method

Let us recall the idea of quadrature for ODE solving.

$$y(t_{l+m}) - y(t_{l+m-1}) = \int_{t_{l+m-1}}^{t_{l+m}} f(t, y(t))dt \tag{5.38}$$

The m-step method now needs to represent the integral by a discrete quadrature scheme over $t_{l+j}, j = 0,1,\ldots,m$ or $j = 0,1,\ldots,m-1$ depending the scheme is implicit or explicit. Following the intuition when we derive quadrature rules, it is first to find a interpolating polynomial.

Let $p(x)$ be a degree $(m-1)$ polynomial interpolating on the nodes $(t_j, f(t_j, y_j))$, $j = l, l+1, \ldots, l+m-1$, then approximately

$$y(t_{l+m}) - y(t_{l+m-1}) = \int_{t_{l+m-1}}^{t_{l+m}} p(t)dt \tag{5.39}$$

Note this is extrapolation. In the next, we try to calculate $p(x)$ with the Newton form backward.

$$\begin{aligned} p(t_{l+m-1} + sh) &= \sum_{k=0}^{m-1} p[t_{l+m-1}, \ldots, t_{l+m-1-k}] \prod_{j=0}^{k-1} (t_{l+m-1} + sh - (t_{l+m-1} - jh)) \\ &= \sum_{k=0}^{m-1} p[t_{l+m-1}, \ldots, t_{l+m-1-k}] h^k \prod_{j=0}^{k-1} (s+j) \end{aligned} \tag{5.40}$$

Therefore

$$y(t_{l+m}) - y(t_{l+m-1}) \approx h \int_0^1 \sum_{k=0}^{m-1} p[t_{l+m-1}, \ldots, t_{l+m-1-k}] h^k \prod_{j=0}^{k-1} (s+j) ds \tag{5.41}$$

This method is called Adam-Bashfort method. The above formula can be written into linear form uniquely. Here are a few examples of Adam-Bashfort m-step methods.

1. $m = 1$. $y_{l+1} - y_l = hf(t_l, y_l)$. This is explicit Euler.
2. $m = 2$. $y_{l+2} - y_{l+1} = \frac{h}{2}(3f(t_{l+1}, y_{l+1}) - f(t_l, y_l))$.
3. $m = 3$. $y_{l+3} - y_{l+2} = \frac{h}{12}(23f(t_{l+2}, y_{l+2}) - 16f(t_{l+1}, y_{l+1}) + 5f(t_l, y_l))$.
4. $m = 4$. $y_{l+4} - y_{l+3} = \frac{h}{24}(55f(t_{l+3}, y_{l+3}) - 59f(t_{l+2}, y_{l+2}) + 37f(t_{l+1}, y_{l+1}) - 9f(t_l, y_l))$.

Now we can study the consistency order of the above m-step method. First the generating polynomial is

$$q(\xi) = \xi^{m-1}(\xi - 1)$$

which satisfies root condition clearly. The local error is to compare the interpolating polynomial and the right-hand-side of (5.41), which means

$$E(t, h) = h\mathcal{O}(h^m) = \mathcal{O}(h^{m+1}) \quad \text{since } \deg(p) = m - 1. \tag{5.42}$$

That means consistency order is $p = m$.

When the Adams method considers implicit scheme (called Adam-Moulton method), that is, the interpolation is including the end point.

$$p(t_j) = f(t_j, y_j), \quad j = l, \ldots, l+m, \tag{5.43}$$

the polynomial is of degree m. The similar approach applies. The consistency order is clearly $p = m+1$ using the error estimate of polynomial interpolation. Here we also list a few cases for this case.

1. $m = 1$. $y_{l+1} - y_l = \frac{h}{2}(f(t_l, y_l) + f(t_{l+1}, y_{l+1}))$. This is Crank-Nicolson method.
2. $m = 2$. $y_{l+2} - y_{l+1} = \frac{h}{12}(5f(t_{l+2}, y_{l+2}) + 8f(t_{l+1}, y_{l+1}) - f(t_l, y_l))$.
3. $m = 3$. $y_{l+3} - y_{l+2} = \frac{h}{24}(9f(t_{l+3}, y_{l+3}) + 19f(t_{l+2}, y_{l+2}) - 5f(t_{l+1}, y_{l+1}) + f(t_l, y_l))$.

5.2 Boundary Value Problem

The boundary value problem (BVP) considers the problem

$$\sum_{j=0}^{m} c_j D^j y = f(x, y, Dy, \dots, D^{m-1}y) \tag{5.44}$$

with boundary conditions

$$\sum_{j=0}^{m-1} c_{kj} D^j y(a) + d_{kj} D^j y(b) = \alpha_k, \quad k = 0, \dots, m-1.$$

All BVP can be formulated into vector form of first order ODE. Let $Y_k = D^k y$, then we can obtain an ODE for the unknown vector $\mathcal{Y} = (Y_0, \dots, Y_m)$ such that

$$DY_k = Y_{k+1}, \quad k = 0, \dots, m-1 \tag{5.45}$$

and

$$c_m DY_m + \sum_{j=0}^{m-1} c_j Y_j = f(x, Y_0, \dots, Y_{m-1}). \tag{5.46}$$

Among all kinds of ODE, the second order ODE is mostly considered (e.g. Sturm-Liouville). We state some properties for these ODEs.

Theorem 5.2.1. *The Sturm-Liouville problem*

$$\begin{aligned} &-y''(x) + r(x)y(x) = f(x), \\ &y(a) = y(b) = 0, \end{aligned} \tag{5.47}$$

where f is a continuous functions. This problem permits a unique solution $y \in C^2([a,b])$ when r is a non-negative continuous function.

Remark 5.2.2. *One can show there exists a unique weak solution $y \in H^1([a,b])$ which is Hölder continuous by embedding, therefore it is a strong solution due to $y'' = ry - f \in C^0([a,b])$.*

5.2.1 Finite Difference Method

The finite difference method is to discretize the derivatives by finite difference. Let us recall the central scheme for first derivative and second derivative of y by

$$\begin{aligned} \frac{y(x+h) - y(x-h)}{2h} &= y'(x) + y'''(\xi)\frac{h^2}{6} \\ \frac{y(x+h) - 2y(x) + y(x-h)}{h^2} &= y''(x) - y^{(4)}(\xi)\frac{h^2}{12} \end{aligned} \tag{5.48}$$

The central scheme usually has better local error than forward or backward difference. In the following, we try to formulate the finite difference method for the Sturm-Liouville problem with $r \geq 0$. Take the grid $\Delta \subset [a,b]$ as

$$\Delta = \{x_k = a + kh,\ k = 0, \dots, N\} \tag{5.49}$$

such that $h = \frac{b-a}{N}$ as step size which stands for the solution resolution. Then we use central scheme for the second derivative at each x_k,

$$\frac{-y_{k+1} + 2y_k - y_{k-1}}{h^2} + r_k y_k = f_k \tag{5.50}$$

where $r_k = r(x_k)$, $f_k = f(x_k)$, $k = 1, \dots, N-1$, the two end points are excluded since they are prescribed as $y_0 = y_N = 0$. This linear system about $\mathcal{Y} = (y_1, \dots, y_{N-1})$ is

$$\frac{1}{h^2}\begin{pmatrix} 2+r_1h^2 & -1 & & & & \\ -1 & 2+r_2h^2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -1 \\ & & & & -1 & 2+r_{N-1}h^2 \end{pmatrix}\mathcal{Y} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix} \tag{5.51}$$

if we denote the right-hand-side as F, then $\mathcal{Y} = A^{-1}F$. There error can be derived from the approximation error of finite difference, which is

$$\frac{1}{h^2}\begin{pmatrix} 2+r_1h^2 & -1 & & & & \\ -1 & 2+r_2h^2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -1 \\ & & & & -1 & 2+r_{N-1}h^2 \end{pmatrix}\widetilde{\mathcal{Y}} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix} + \mathcal{O}(h^2)\|y^{(4)}\|_\infty \tag{5.52}$$

where $\widetilde{\mathcal{Y}}$ is the exact solution evaluated at nodes x_k. Therefore we can find the numerical error by

$$\mathcal{Y} - \widetilde{\mathcal{Y}} = A^{-1}\left(\mathcal{O}(h^2)\|y^{(4)}\|_\infty\right) \tag{5.53}$$

For reliable numerical solution, we will require $\|\mathcal{Y} - \widetilde{\mathcal{Y}}\|_\infty \to 0$ as $h \to 0$. This will need the estimate of $\|A\|_\infty$.

Lemma 5.2.3. *Let $0 \le S \le T$, that is, $0 \le s_{ij} \le t_{ij}$ for all elements. Then*

$$\|S\|_\infty \le \|T\|_\infty.$$

5.2.2 Galerkin Method

5.3 Exercises

5.3.1 Theoretical Part

Problem 5.3.1. *Show the Runge Kutta RK4 has order of convergence $p = 4$.*

Problem 5.3.2. *Prove Lemma 5.1.24.*

Problem 5.3.3. *Determine the consistency order of the multi-step method*

$$y_{l+2} - y_l = \frac{h}{3}\left(f(t_{l+2}, y_{l+2}) + 4f(t_{l+1}, y_{l+1}) + f(t_l, y_l)\right) \tag{5.54}$$

Problem 5.3.4. *For what values of γ, the following method*

$$y_{l+3} + \gamma(y_{l+2} - y_{l+1}) - y_l = \frac{(3+\gamma)h}{2}\left(f(t_{l+2}, y_{l+2}) + f(t_{l+1}, y_{l+1})\right) \tag{5.55}$$

provides a convergent method, what is the order.

5.3.2 Computational Part

Problem 5.3.5. *Implement Runge-Kutta method RK4 and apply the method for the ODE*

$$y' = -y, \quad y(0) = 1. \tag{5.56}$$

Validate the convergence order on $[0, 1]$.

Problem 5.3.6. *Implement the Adam-Bashfort method for $m = 2$. The initial two steps can be computed first by RK4.*

Chapter 6

Partial Differential Equations

6.1 Finite Difference Method

6.2 Finite Volume Method

6.3 Pseudo Spectral Method

6.4 IMEX Method

6.5 Sector Operators

6.6 PINN

Chapter 7

Integral Equations

7.1 Nyström Method

7.2 Galerkin Method

Chapter 8

Matrix Computation

8.1 Factorization

8.2 Preconditioning

8.3 Large-scale Computation

Chapter 9

Extended Topics

9.1 Radial Basis Interpolation

9.2 Quadrature in multi-dimension

9.3 Stiff ODE

9.4 Fast Multipole Method

9.5 Random Algorithm

9.6 Optimization

9.7 Sampling Algorithm

9.8 Parallel Computing

9.9 Regularization

9.10 Optimal Transport

9.11 Point Cloud

9.12 Inverse Crime

9.13 Low Rank Approximation

9.14 Finite Element Method

9.15 Computation on Graph