MATH 6651 Advanced Numerical Methods

Joe Tran

November 20, 2025

Preface

These are the first edition of these lecture notes for MATH 6651 (Advanced Numerical Methods). Consequently, there may be several typographical errors, missing exposition on necessary background, and more advanced topics for which there will not be time in class to cover. Future iterations of these notes will hopefully be fairly self-contained provided one has the necessary background. If you come across any typos, errors, omissions, or unclear expositions, please feel free to contact me so that I may continually improve these notes.

Contents

Chapter 1

Optimization Methods

Optimization lies at the heart of scientific computing, engineering, and applied mathematics.

The fundamental task is to minimize (or maximize) an objective function $f(x)$ subject to possible constraints.

In this chapter, we focus on core optimization methods that provide the building blocks for more advanced algorithms.

We begin with *search methods in one dimension*, where the goal is to locate a minimum of a unimodal function defined on an interval. Techniques such as the **golden-section method** and the **Fibonacci method** provide efficient strategies for interval reduction without requiring derivative information. These methods illustrate how mathematical structure, such as the golden ratio or Fibonacci recursion, can yield optimal contraction rates.

Next, we study the *simplex method in two dimensions*. Here, the feasible region is a polygon defined by linear inequalities, and the minimum of a linear function occurs at one of its vertices. The simplex method systematically moves between vertices, providing a geometric and algorithmic framework for linear programming in higher dimensions.

Finally, we extend to *multivariate optimization in* $\mathbb{R}^n$. We consider both first-order and second-order approaches: the **steepest descent method**, which follows the direction of steepest decrease in the objective function, and **Newton's method**, which exploits curvature information from the Hessian to achieve faster convergence. These methods highlight the trade-offs between simplicity, computational cost, and convergence speed.

1.1 Search Methods in One-Dimension

In one-dimensional optimization, the task is to minimize a function $f : [a,b] \to \mathbb{R}$ on a closed interval without assuming the availability of derivative information. If the function is unimodal, the minimizer can be localized by successively shrinking the interval. The golden-section and Fibonacci methods provide efficient strategies to achieve this, guaranteeing convergence with minimal function evaluations. These classical interval-reduction techniques are widely used when derivative-free optimization is required.

1.1.1 The Golden Method

The golden-section method is a derivative-free search technique for finding the minimum of a unimodal function $f : [a,b] \to \mathbb{R}$. It belongs to the class of *interval reduction methods*, where the interval containing the minimizer is successively contracted while ensuring that the minimizer remains inside.

Definition 1.1.1.1. Suppose $f \in C[a,b]$ and there exists a $c \in (a,b)$ such that

$$f(c) < f(a) \quad \text{and} \quad f(c) < f(b)$$

Then f has at least one local minimizer $x^* \in (a,b)$ with

$$f(x^*) = \min_{x \in [a,b]} f(x)$$

The idea is instead of derivatives, we shrink the interval step-by-step by comparing values of $f(x)$ at certain test points.

At the kth iteration, the current interval is $[a_k, b_k]$. Choose interior points.

$$\begin{aligned} c_k &= b_k - \rho(b_k - a_k) \\ d_k &= a_k + \rho(b_k - a_k) \end{aligned}$$

with parameter $\frac{1}{2} < \rho < 1$. Then compare function values $f(c_k)$ and $f(d_k)$.

- If $f(c_k) < f(d_k)$, then set $a_{k+1} = a_k$ and $b_{k+1} = d_k$, so $[a_{k+1}, b_{k+1}] = [a_k, d_k]$.

- If $f(c_k) > f(d_k)$, then set $a_{k+1} = c_k$ and $b_{k+1} = b_k$, so $[a_{k+1}, b_{k+1}] = [c_k, b_k]$.

Remark 1.1.1.2. We only consider $\rho \in \left(\frac{1}{2}, 1\right)$. If it were the case that $\rho = \frac{1}{2}$, then we have

$$\begin{aligned} c_k &= b_k - \frac{1}{2}(b_k - a_k) = \frac{a_k + b_k}{2} \\ d_k &= a_k + \frac{1}{2}(b_k - a_k) = \frac{a_k + b_k}{2} \end{aligned}$$

In other words, c_k and d_k will collapse to the same point. Thus, this results in sampling only at the midpoint, and cannot compare two function values to decide which subinterval contains the minimizer.

On the other hand, if $\rho = 1$, then

$$\begin{aligned} c_k &= b_k - (b_k - a_k) = a_k \\ d_k &= a_k + (b_k - a_k) = b_k \end{aligned}$$

In this case, only comparing endpoints a_k and b_k, which we already know from the start.

If ρ was arbitrarily chosen, then every time we compute two new points c_k and d_k, we need to evaluate $f(c_k)$ and $f(d_k)$, so two function evaluations per iteration.

But there is a question that we would like to ask ourselves: What if we can reuse one of these function evaluations from the previous step?

We want the interval lengths to satisfy

(I) $I_k = b_k - a_k$

(II) $\frac{I_k}{I_{k+1}} = R$ for all k

Additionally, we impose the following condition

(III) $I_k = I_{k+1} + I_{k+2}$

This is because if the overlap pattern is consistent, then one of the old test points becomes a new test point in the next step.

From the conditions, we have

$$\frac{I_{k+1}}{I_k} = \frac{1}{R} \implies I_{k+1} = \frac{1}{R} I_k$$

and

$$\frac{I_{k+2}}{I_k} = \frac{I_{k+2}}{I_{k+1}} \cdot \frac{I_{k+1}}{I_k} = \frac{1}{R^2} \implies I_{k+2} = \frac{1}{R^2} I_k$$

Then substitute to (III) to obtain

$$\begin{aligned} I_k &= \frac{1}{R} I_k + \frac{1}{R^2} I_k \\ 1 &= \frac{1}{R} + \frac{1}{R^2} \\ 0 &= R^2 - R - 1 \end{aligned}$$

Then using the quadratic formula, we have

$$R = \frac{1+\sqrt{5}}{2} \approx 1.618$$

which is precisely the Golden ratio $\varphi = 1.618$. In particular, the value of ρ we can obtain is then

$$\rho = \frac{1}{R} = \frac{1}{\varphi} \approx 0.618$$

At each step then

$$\begin{aligned} c_k &= b_k - 0.618(b_k - a_k) \\ d_k &= a_k + 0.618(b_k - a_k) \end{aligned}$$

Such value of $R = \frac{1+\sqrt{5}}{2}$ is the most optimal value to use, and this method is known as the Golden Method.

Remark 1.1.1.3. Such a value $\rho = 0.618$ is the most optimal value to use in the search method, rather than, for example $\rho = \frac{2}{3}$. In particular $\rho = \frac{2}{3}$ is not optimal since with such a ρ, each new interval keeps one of the interior points and discards the other. However, in the next iteration, you have to compute both new interior points again, and you cannot reuse the one from before. This means two new function evaluations per step. So while it converges, it wastes effort.

Theorem 1.1.1.4 (Error Bound Estimate of Golden Method). *After k iterations:*

$$|p_{k+1} - x^*| \leq \frac{1}{2}\left(\frac{1}{R}\right)^k (b-a)$$

Since $\frac{1}{R} \approx 0.618$, the error shrinks like 0.618^k. That is exponential convergence!

At the end, instead of just taking the midpoint p_{k+1}, we can interpolate a quadratic polynomial $P_2(x)$ through three points, say for example, a_{k+1}, c_k, and b_{k+1}.

- Then find the minimizer of $P_2(x)$ by solving $P_2'(x) = 0$.
- This gives a sharper approximation.

Example 1.1.1.5. We apply the Golden Method on $f(x) = 2\sin(2x) + 2$ on $[a_1, b_1] = [1.5, 3.1]$ with $\rho = \frac{\sqrt{5}-1}{2} \approx 0.618$ as follows:

Step 1: Find c_1 and d_1 and compare function values. We have

$$\begin{aligned} c_1 &= b_1 - 0.618(b_1 - a_1) = 3.1 - 0.618(3.1 - 1.5) \approx 2.111 \\ d_1 &= a_1 + 0.618(b_1 - a_1) = 1.5 + 0.618(3.1 - 1.5) \approx 2.489 \end{aligned}$$

Then computing the function values,

$$\begin{aligned} f(c_1) &= 2\sin(2 \cdot 2.111) + 2 \approx 0.235 \\ f(d_1) &= 2\sin(2 \cdot 2.489) + 2 \approx 0.070 \end{aligned}$$

Since $f(c_1) > f(d_1)$, we keep the right part, so $a_2 = c_1$ and $b_2 = b_1$, and thus,

$$[a_2, b_2] = [2.111, 3.1]$$

Step 2: Find c_2 and d_2 and compare function values. We have

$$\begin{aligned} c_2 &= b_2 - 0.618(b_2 - a_2) = 3.1 - 0.618(3.1 - 2.111) \approx 2.489 \\ d_2 &= a_2 + 0.618(b_2 - a_2) = 2.111 + 0.618(3.1 - 2.111) \approx 2.722 \end{aligned}$$

Then computing the function values,

$$\begin{aligned} f(c_1) &= 2\sin(2 \cdot 2.489) + 2 \approx 0.070 \\ f(d_1) &= 2\sin(2 \cdot 2.722) + 2 \approx 0.513 \end{aligned}$$

Since $f(c_2) < f(d_2)$, we keep the left part, so $a_3 = a_2$ and $b_3 = d_2$, and thus,

$$[a_3, b_3] = [2.111, 2.722]$$

Step 3: Find c_3 and d_3 and compare function values. We have

$$\begin{aligned} c_3 &= b_3 - 0.618(b_3 - a_3) = 2.722 - 0.618(2.722 - 2.111) \approx 2.345 \\ d_3 &= a_3 + 0.618(b_3 - a_3) = 2.111 + 0.618(2.722 - 2.111) \approx 2.489 \end{aligned}$$

Then computing the function values,

$$\begin{aligned} f(c_3) &= 2\sin(2 \cdot 2.345) + 2 \approx 5.39 \times 10^{-4} \\ f(d_3) &= 2\sin(2 \cdot 2.489) + 2 \approx 0.070 \end{aligned}$$

Since $f(c_3) < f(d_3)$, we keep the left part, so $a_4 = a_3$ and $b_4 = d_3$, and thus,

$$[a_4, b_4] = [2.111, 2.489]$$

From here, we can estimate where the minimizer is located at given the 4th step

$$p_4 = \frac{a_4 + b_4}{2} = \frac{2.111 + 2.489}{2} \approx 2.300$$

Golden Method Code: `golden_method.py`

1.1.2 The Fibonacci Method

The Fibonacci method is another interval reduction technique for finding the minimum of a unimodal function in one dimension. Like the Golden method, it works by comparing function values at two interior points c_k and d_k of the interval $[a_k, b_k]$ and discarding the half that cannot contain the minimizer.

What makes the Fibonacci method distinctive is that the interval lengths are chosen according to ratios of Fibonacci numbers, rather than the fixed golden ratio. This design ensures that after a predetermined number of steps n, the interval length is minimized in the sense that c_{n-1} and d_{n-1} coincide, giving an approximation of the minimizer with guaranteed accuracy.

The key recurrence condition is

$$I_k = I_{k+1} + I_{k+2}$$

for all $k \geq 1$ with $I_k = b_k - a_k$. Using properties of the Fibonacci sequence, this yields precise interval shrinkage formulas and a stopping criterion directly tied to the chosen number of iterations.

In practice, the Fibonacci method is especially effective when the number of iterations can be fixed in advance, offering a more optimal use of function evaluations compared to the Golden method.

Recall that the Fibonacci sequence is defined as follows: $F_0 = 1$, $F_1 = 1$, and for all $j \geq 0$,

$$F_{j+2} = F_j + F_{j+1}$$

To construct the step, let $I_1 = b - a$ be the initial interval length. For $k \geq 2$, define

$$\frac{I_k}{I_1} = \frac{F_{n-k+1}}{F_n}$$

where n is the predetermined number of iterations. Then

$$c_k = b_k - \frac{I_{k+1}}{I_1} I_1$$
$$d_k = a_k + \frac{I_{k+1}}{I_1} I_1$$

At each step, as with the Golden Method

- If $g(c_k) < g(d_k)$, then $a_{k+1} = a_k$ and $b_{k+1} = d_k$.
- Otherwise, $a_{k+1} = c_k$ and $b_{k+1} = b_k$.

The algorithm terminates at step n, where $c_{n-1} = d_{n-1}$. The final approximation of the minimizer is

$$p_{n-1} = c_{n-1} = d_{n-1}$$

Alternatively, at earlier steps

$$p_{k+1} = \frac{1}{2}(a_{k+1} + b_{k+1})$$

can serve as an approximation once the interval is sufficiently small.

Example 1.1.2.1. For $n = 7$, we have

$$\frac{I_2}{I_1} = \frac{F_6}{F_7} = \frac{13}{21}$$
$$\frac{I_3}{I_1} = \frac{F_5}{F_7} = \frac{8}{21}$$
$$\frac{I_4}{I_1} = \frac{F_4}{F_7} = \frac{5}{21}$$
$$\frac{I_5}{I_1} = \frac{3}{21}$$
$$\frac{I_6}{I_1} = \frac{2}{21}$$
$$\frac{I_7}{I_1} = \frac{1}{21}$$

Example 1.1.2.2. We apply the Fibonacci method on $f(x) = 2\sin(2x) + 2$ on $[a_1, b_1] = [1.5, 3.1]$ with $n = 5$. Note that

$$I_1 = 3.1 - 1.5 = 1.6 = \frac{8}{5}$$

and let $\{F_j\}_{j=0}^{\infty}$ denote the Fibonacci sequence.

Step 1: We calculate $\frac{I_k}{I_1} = \frac{F_{n-k+1}}{F_n}$ where $n = 5$ and $k = 2, 3, 4, 5$. Indeed, we have

$$\frac{I_2}{I_1} = \frac{F_4}{F_5} = \frac{5}{8}$$
$$\frac{I_3}{I_1} = \frac{F_3}{F_5} = \frac{3}{8}$$
$$\frac{I_4}{I_1} = \frac{F_2}{F_5} = \frac{2}{8}$$
$$\frac{I_5}{I_1} = \frac{F_1}{F_5} = \frac{1}{8}$$

Step 2: Compute c_1 and d_1 and compare function values.

We have

$$c_1 = b_1 - \left(\frac{I_2}{I_1}\right) I_1 = 3.1 - \frac{5}{8} \cdot \frac{8}{5} = 2.1$$
$$d_1 = a_1 + \left(\frac{I_2}{I_1}\right) I_1 = 1.5 + \frac{5}{8} \cdot \frac{8}{5} = 2.5$$

Then computing the function values

$$f(c_1) \approx 0.257 \quad f(d_1) \approx 0.082$$

Since $f(c_1) > f(d_1)$, then we keep the right part, and set $a_2 = c_1 = 2.1$ and $b_2 = b_1 = 3.1$, so $[a_2, b_2] = [2.1, 3.1]$.

Step 3: Compute c_2 and d_2 and compare function values.

We have

$$c_2 = b_2 - \left(\frac{I_3}{I_1}\right) I_1 = 3.1 - \frac{3}{8} \cdot \frac{8}{5} = 2.5$$
$$d_2 = a_2 + \left(\frac{I_3}{I_1}\right) I_1 = 2.1 + \frac{3}{8} \cdot \frac{8}{5} = 2.7$$

Then computing the function values

$$f(c_2) \approx 0.082 \quad f(d_2) \approx 0.454$$

Since $f(c_2) < f(d_2)$, then we keep the left part, and set $a_3 = a_2 = 2.1$ and $b_3 = d_2 = 2.7$, so $[a_3, b_3] = [2.1, 2.7]$.

Step 4: Compute c_3 and d_3 and compare function values.

We have

$$c_3 = b_3 - \left(\frac{I_4}{I_1}\right) I_1 = 2.7 - \frac{2}{8} \cdot \frac{8}{5} = 2.3$$
$$d_3 = a_3 + \left(\frac{I_4}{I_1}\right) I_1 = 2.1 + \frac{2}{8} \cdot \frac{8}{5} = 2.5$$

Then computing the function values

$$f(c_3) \approx 0.013 \quad f(d_3) \approx 0.082$$

Since $f(c_3) < f(d_3)$, then we keep the left part, and set $a_4 = a_3 = 2.1$ and $b_4 = d_3 = 2.5$, so $[a_4, b_4] = [2.1, 2.5]$.

Step 5: Compute c_4 and d_4 and stop.

When computed, we get $c_4 = d_4 = 2.3$.

Step 6: Compute p_4, we have

$$p_4 = \frac{a_4 + b_4}{2} = \frac{2.1 + 2.5}{2} = 2.3$$

Remark 1.1.2.3. The Fibonacci method requires knowing in advance how many iterations n will be performed.

Remark 1.1.2.4. Compare to the Golden method, it achieves slightly more efficient interval reduction for the same number of function values.

Remark 1.1.2.5. If the number of iterations is not predetermined, the Golden method is more convenient to implement.

Below shows the algorithm (pseudocode) of the Fibonacci method.

- Input: Interval $[a, b]$; Objective function $g(x)$; maximum steps n; tolerance T.
- Output: approximation p to the minimizer.

```
Step 1. Set I1 = b - a.
        Compute Fibonacci numbers {Fj} up to Fn.

Step 2. Initialize k = 1.
Step 3. For k <= n - 1 do Steps 4-8:

    Step 4. Compute ratio:
            r = F_{n - k + 1} / Fn.

    Step 5. Compute points:
            c = b - r * I1,
            d = a + r * I1.

    Step 6. If g(c) < g(d) then
                b = d
            else
                a = c

    Step 7. If (1/2)(b - a) < Tol then
                p = (a + b)/2
                Output p and STOP.

    Step 8. Set k = k + 1

Step 9. Final approximation:
    p = (a + b)/2
    or p = c = d (when k = n - 1)

Output p.
```

Fibonacci Method Code: fibonacci_method.py

1.2 Simplex Method in Two Dimensions

Optimization in two variables introduces new geometric and algorithmic ideas. Instead of shrinking intervals as in one-dimensional search, we now approximate the minimizer (x^*, y^*) of a function $g : \mathbb{R}^2 \to \mathbb{R}$ by iteratively refining a simplex in two dimensions, a triangle.

Definition 1.2.1 ((**Minimization Problem in 2D**). Given a continuous objective function $g(x, y)$ defined on a rectangular domain

$$\Omega = [x_l, x_r] \times [y_b, y_t] \subseteq \mathbb{R}^2$$

the goal is to compute

$$g(x^*, y^*) = \min_{(x,y)\in\Omega} g(x, y)$$

where (x^*, y^*) is the local minimizer of g.

The *simplex method* iteratively refines a simplex (in 2D, a triangle) $\Delta a_k b_k c_k$ by reflection or contraction of the previous simplex $\Delta a_{k-1} b_{k-1} c_{k-1}$ so that

1. The new simplex is closer to (x^*, y^*),
2. The area $S(\Delta a_k b_k c_k)$ shrinks toward zero.

The centroid

$$p_k = \frac{1}{3}(a_k + b_k + c_k)$$

serves as the current approximation to the minimizer.

At each iteration, we label the vertices of Δabc by their function values:

- a: the worst (highest function value, $g(a)$ largest),
- b: the second highest.
- c: the best (lowest).

The algorithm then attempts to replace a with a better candidate point, using reflection or contraction.

Let M be the midpoint of $[bc]$. The reflection point a' is defined by

$$a' = a + 2(M - a)$$

that is, a' is the reflection of a across M. Depending on the values of $g(a')$, $g(b)$, and $g(c)$, we proceed by cases.

Case 1. If $g(a') < g(b)$ (and hence, $g(a') < g(a)$), accept a': new simplex $\Delta ba'c$.

Case 2. If $g(a') < g(c)$, check expansion. Let a'' be the expansion point (further along the reflection direction). If $g(a'') < g(b)$, accept a''; else, accept a'.

Case 3. If $g(a') \geq g(a)$, contract a towards M to a^*. If $g(a^*) < \min\{g(a), g(b)\}$, accept a^*.

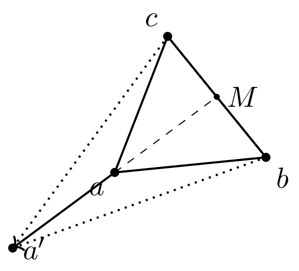


Figure 1.1: Case 1: reflect a across M to a' and set new simplex $\triangle a'bc$.

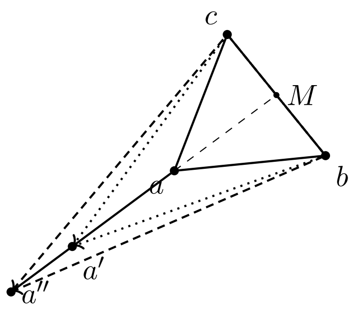


Figure 1.2: Case 2: if reflection is very good, try expansion to a''; accept a'' if it improves, else keep a'.

Case 4. If $g(b) \le g(a') < g(a)$, contract a' toward M to a^{**}. If $g(a^{**}) < \min\{g(a), g(b)\}$, accept a^{**}.

Case 5. If none of the above apply, contract a and b towards c to $\bar{a}, \bar{b}$, and set new simplex $\Delta\bar{a}\bar{b}c$.

Proposition 1.2.2. *In the simplex method in 2D, given Δabc such that*

- *a is the worst point,*
- *b is the second best point,*
- *c is the best point,*

then

1. *Reflection: $a' = a + 2(M - a)$,*
2. *Expansion: $a'' = a + 3(M - a)$,*

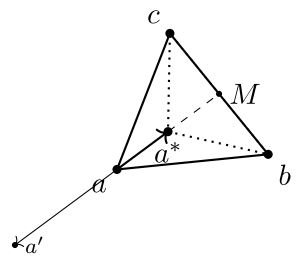


Figure 1.3: Case 3: reflection is not better; contract a toward M to a^* and update.

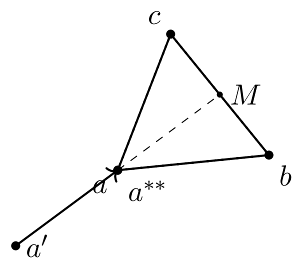


Figure 1.4: Case 4: reflection is better than a but not than b; contract a' toward M to a^{**}.

3. *Contraction of a towards M to a^*:* $a^* = a + \frac{1}{2}(M - a)$,

4. *Contraction of a' towards M to a^{**}:* $a^{**} = a' + \frac{1}{2}(M - a)$,

5. *Contraction of a and b towards c to $\bar{a}$ and $\bar{b}$:* $\bar{a} = \frac{1}{2}(a + c)$, $\bar{b} = \frac{1}{2}(b + c)$.

Proof. To see that (1) holds, reflect a across M, i.e. M is the midpoint of $\overline{aa'}$, i.e. $\overrightarrow{aa'} = 2\overrightarrow{aM}$ so

$$a' = a + 2(M - a) = 2M - a$$

To see that (2) holds, expand further in the same direction than reflection. A convenient choice is $\overrightarrow{aa''} = 3\overrightarrow{aM}$, so

$$a'' = a + 3(M - a) = 3M - 2a$$

To see that (3) holds, move halfway from a to M, i.e. $\overrightarrow{aa^*} = \frac{1}{2}\overrightarrow{aM}$, thus,

$$a^* = a + \frac{1}{2}(M - a) = \frac{1}{2}(a + M)$$

To see that (4) holds, move halfway from a' to M, i.e. $\overrightarrow{a'a^{**}} = \frac{1}{2}\overrightarrow{a'M}$, thus,

$$a^{**} = a' + \frac{1}{2}(M - a') = \frac{1}{2}(a' + M)$$

To see that (5) holds, shrink the vertices a and b toward the best vertex c by taking midpoints with c

$$\bar{a} = \frac{1}{2}(a + c), \quad \bar{b} = \frac{1}{2}(b + c).$$

$\square$

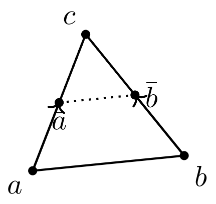


Figure 1.5: Case 5: shrink the simplex toward the best vertex c: $\bar{a} = c + \sigma(a - c)$, $\bar{b} = c + \sigma(b - c)$ (e.g., $\sigma = \frac{1}{2}$).

The method is terminated if either:

1. The maximum iteration number N is reached, or
2. The area $S(\Delta a_k b_k c_k)$ satisfies $S(\Delta a_k b_k c_k) < \text{Tol}$.

The Heron formula is used to compute the area.

Below describes the algorithm for the simplex method in $\mathbb{R}^2$.

1. Input: initial simplex $\triangle a_0 b_0 c_0$, objective $g(x, y)$, max iterations N, tolerance Tol.
2. For $k = 1, 2, \ldots, N$:
   (a) Label vertices a, b, c by worst $\to$ best.
   (b) Generate trial point(s) $a', a'', a^*, a^{**}, \bar{a}, \bar{b}$ as required.
   (c) Update the simplex using Cases 1–5.
   (d) If area $S(\triangle a_k b_k c_k) < \text{Tol}$, stop and output $p_k = \frac{1}{3}(a_k + b_k + c_k)$.

Example 1.2.3. Consider the minimization of function

$$g(x, y) = -(3x^2 + 2xy + y^2)e^{-x^2-3y^2}$$

Starting from the initial triangle $a_0(0, 0)$, $b_0(0.2, 0)$, and $c_0(0, 0.2)$, suppose we want to find $\Delta a_2 b_2 c_2$ using the simplex method.

Step 1. We first compute $g(a_0)$, $g(b_0)$, and $g(c_0)$, and compare. Indeed, we find that

$$g(a_0) = 0, \quad g(b_0) = -0.115, \quad g(c_0) = -0.035$$

and so $g(a_0) > g(c_0) > g(b_0)$. In this case, we have

- a_0 is the highest point.
- c_0 is the second highest point.
- b_0 is the lowest point.

Step 2. Next, we find the midpoint of $[b_0c_0]$, and then find the point a', and then compute $g(a')$, and then compare with $g(c_0)$ and $g(a)$. In this case, we have

$$M(x_M, y_M) = \left(\frac{1}{2}(0.2+0), \frac{1}{2}(0+0.2)\right) = (0.1, 0.1)$$

and

$$\begin{pmatrix} x_{a'} \\ y_{a'} \end{pmatrix} = \begin{pmatrix} x_a + 2(x_M - x_a) \\ y_a + 2(y_M - y_a) \end{pmatrix} = \begin{pmatrix} 0 + 2(0.1 - 0) \\ 0 + 2(0.1 - 0) \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}$$

So we have $a'(0.2, 0.2)$. Then

$$g(a') = -0.205$$

Comparing function values, we see that $g(a') < g(c_0) < g(a)$ and compare $g(a') < g(b_0)$. In this case, a' cannot be accepted.

Step 3. Now we check $a''(x_{a''}, y_{a''})$, which is given by

$$\begin{pmatrix} x_{a''} \\ y_{a''} \end{pmatrix} = \begin{pmatrix} x_a + 3(x_M - x_a) \\ y_a + 3(y_M - y_a) \end{pmatrix} = \begin{pmatrix} 0 + 3(0.1 - 0) \\ 0 + 3(0.1 - 0) \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.3 \end{pmatrix}$$

so $a''(0.3, 0.3)$, and thus $g(a'') = -0.377$. Comparing function values of a'' and c_0, we have $g(a'') < g(c_0)$, so by Case 2, a'' is accepted, so we have the new simplex $\Delta c_0 a'' b_0$.

Step 4. Now for the new simplex, we have $g(a'') < g(b_0) < g(c_0)$. In this case, we have

- c_0 is the highest point.
- b_0 is the second highest point.
- a'' is the lowest point.

Step 5. Next, we compute the midpoint $N(x_N, y_N)$ of $[a''b_0]$. In this case, we have

$$(x_N, y_N) = \left(\frac{1}{2}(0.2+0.3), \frac{1}{2}(0+0.3)\right) = (0.25, 0.15)$$

Now, we get the point $c'(x_{c'}, y_{c'})$, where

$$\begin{pmatrix} x_{c'} \\ y_{c'} \end{pmatrix} = \begin{pmatrix} x_{c_0} + 2(x_N - x_{c_0}) \\ y_{c_0} + 2(y_N - y_{c_0}) \end{pmatrix} = \begin{pmatrix} 0 + 2(0.25 - 0) \\ 0.2 + 2(0.15 - 0.2) \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}$$

So we have $c'(0.5, 0.1)$. Then

$$g(c') = -0.647$$

Comparing function values, we see that

$$g(c') < g(a'') < g(b_0) < g(c_0)$$

So Case 1 cannot be accepted.

Step 6. Next, try the point c'':

$$\begin{pmatrix} x_{c'} \\ y_{c'} \end{pmatrix} = \begin{pmatrix} x_{c_0} + 3(x_N - x_{c_0}) \\ y_{c_0} + 3(y_N - y_{c_0}) \end{pmatrix} = \begin{pmatrix} 0 + 3(0.25 - 0) \\ 0.2 + 3(0.15 - 0.2) \end{pmatrix} = \begin{pmatrix} 0.75 \\ 0.05 \end{pmatrix}$$

Then computing function value, $g(c'') = -0.998$, and comparing $g(c'')$ with $g(b_0)$, we have $g(c'') < g(b_0)$, so c'' is accepted. By Case 2, we get the new simplex $\Delta a''b_0c''$.

Step 7. Now we find the approximation minimum point p_2. Indeed,

$$p_2 = \frac{1}{3}(a'' + b_0 + c'') = \left(\frac{5}{12}, \frac{7}{60}\right)$$

Remark 1.2.4.

- **Advantages:** (i) derivative-free, (ii) guaranteed convergence in $\mathbb{R}^2$.
- **Disadvantages:** (i) slow convergence, (ii) case distinctions and geometry become more complicated in higher dimensions.

Simplex Method Code: `simplex_method.py`

1.3 Steepest Descent Method and Newton's Method

Up to now, we have studied optimization methods in low dimensions: search strategies in one dimension (Golden and Fibonacci) and the simplex method in two dimensions. In this section, we turn to optimization in higher dimensions, where the objective function is

$$g(\mathbf{x}), \qquad \mathbf{x} \in \Omega \subset \mathbb{R}^n.$$

The goal is to approximate a local minimizer

$$g(\mathbf{x}^*) = \min_{\mathbf{x}\in\Omega} g(\mathbf{x}).$$

Optimization in $\mathbb{R}^n$ is fundamentally different from the 1D and 2D cases: instead of shrinking an interval or contracting a triangle, we must move iteratively along directions in n-dimensional space. The general template is a *line search method:*

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{z}^{(k)}$$

where $\mathbf{z}^{(k)}$ is a search direction and $\alpha_k > 0$ is the step length chosen to minimize g along that line.

1.3.1 Steepest Descent Method

The most natural choice of direction is the negative gradient:

$$\mathbf{z}^{(k)} = -\nabla g(\mathbf{x}^{(k-1)}) = \begin{bmatrix} -\frac{\partial g}{\partial x_1}(\mathbf{x}^{(k-1)}) \\ -\frac{\partial g}{\partial x_2}(\mathbf{x}^{(k-1)}) \\ \vdots \\ -\frac{\partial g}{\partial x_n}(\mathbf{x}^{(k-1)}) \end{bmatrix}$$

which points in the direction of steepest decrease of g. At each iteration, a line search determines α_k that approximately minimizes

$$h(\alpha) = g(\mathbf{x}^{(k-1)}) + \alpha \mathbf{z}^{(k)}$$

This is the *steepest descent method*. It is simple, requires only first derivatives, and always converges, but convergence can be very slow in practice, especially for ill-conditioned problems.

Example 1.3.1.1. Consider the quadratic function in two variables

$$g(x, y) = -4x - 2y - x^2 + 2x^4 + 3y^2$$

We want to minimize $g(x, y)$ using steepest descent.

Step 0: Start from $\mathbf{x}^{(0)} = (0,0)^T$. Compute the gradient:

$$\nabla g(x,y) = \begin{bmatrix} -4 - 2x + 8x^3 \\ -2 + 6y \end{bmatrix}$$

At $\mathbf{x}^{(0)}$, we have

$$\nabla g(0,0) = \begin{bmatrix} -4 \\ -2 \end{bmatrix}$$

Step 1: Search direction (normalized):

$$\mathbf{z}^{(1)} = -\frac{\nabla g(0,0)}{\|\nabla g(0,0)\|} = \frac{1}{\sqrt{20}} \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix}$$

Step 2: (Line Search) Define $h(\alpha) = g(\mathbf{x}^{(0)} + \alpha \mathbf{z}^{(1)})$. Approximate α^* using quadratic interpolation with $\alpha = 0, \frac{1}{2}, 1$.

- $h(0) = g(0,0) = 0$.
- $h\left(\frac{1}{2}\right) = g\left(\frac{1}{\sqrt{5}}, \frac{1}{2\sqrt{5}}\right) = -2.206$.
- $h(1) = g\left(\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right) = -3.392$.

Interpolation gives $\alpha^* = 1.331$.

Step 3: (Update)

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha^* \mathbf{z}^{(1)} = \begin{bmatrix} 1.190 \\ 0.595 \end{bmatrix}$$

Thus, the first step moves significantly toward the minimizer. Further iterations continue in the steepest descent direction at each point, but convergence is only linear and may zig-zag in narrow valleys.

Remark 1.3.1.2. We can get a quick, decent step-length by quadratic interpolation using three samples of the 1D line search function

$$\varphi(\alpha) = g(\mathbf{x}^{(k-1)} + \alpha \mathbf{z}^{(k)})$$

at $\alpha \in \left\{0, \frac{1}{2}, 1\right\}$. Here is another simple derivation that we can use for efficiency. Fit $\varphi(\alpha) \approx a\alpha^2 + b\alpha + c$, then take the parabola's minimizer $\alpha^* = -\frac{b}{2a}$.

Let $\varphi_0 = \varphi(0)$, $\varphi_{\frac{1}{2}} = \varphi\left(\frac{1}{2}\right)$, and $\varphi_1 = \varphi(1)$. Solving for a, b and c, and plugging into $-\frac{b}{2a}$, gives us the handy formula

$$\alpha^* = \frac{3\varphi_0 + \varphi_1 - 4\varphi_{\frac{1}{2}}}{4(\varphi_1 - 2\varphi_{\frac{1}{2}} + \varphi_0)}$$

with $a = 2\varphi_1 - 4\varphi_{\frac{1}{2}} + 2\varphi_0$, $b = 4\varphi_{\frac{1}{2}} - 3\varphi_0 - \varphi_1$, and $c = \varphi_0$. To see this, define

$$q(\alpha) = a\alpha^2 + b\alpha + c$$

We want $q(0) = \varphi_0$, $q\left(\frac{1}{2}\right) = \varphi_{\frac{1}{2}}$, and $q(1) = \varphi_1$. Indeed, note

- If $\alpha = 0$, then $q(0) = c = \varphi_0$.
- If $\alpha = \frac{1}{2}$, then $q\left(\frac{1}{2}\right) = \frac{1}{4}a + \frac{1}{2} + c = \varphi_{\frac{1}{2}}$.
- If $\alpha = 1$, then $q(1) = a + b + c = \varphi_1$

Since $c = \varphi_0$, then rewrite the other two: From $\alpha = 1$

$$a + b + \varphi_0 = \varphi_1 \implies a + b = \varphi_1 - \varphi_0$$

From $\alpha = \frac{1}{2}$,

$$\frac{1}{4}a + \frac{1}{2}b + \varphi_0 = \varphi_{\frac{1}{2}} \implies \frac{1}{4}a + \frac{1}{2}b = \varphi_{\frac{1}{2}} - \varphi_0$$

Multiply the last equation by 4 so that

$$a + 2b = 4(\varphi_{\frac{1}{2}} - \varphi_0)$$

Now we solve

$$\begin{cases} a + b = \varphi_1 - \varphi_0 \\ a + 2b = 4(\varphi_{\frac{1}{2}} - \varphi_0) \end{cases}$$

Subtract the first equation from the second to obtain

$$b = 4\varphi_{\frac{1}{2}} - 3\varphi_0 - \varphi_1$$

Then

$$a = \varphi_1 - \varphi_0 - b = 2\varphi_1 - 4\varphi_{\frac{1}{2}} + 2\varphi_0$$

And thus,

$$\alpha^* = -\frac{b}{2a} = \frac{3\varphi_0 + \varphi_1 - 4\varphi_{\frac{1}{2}}}{4(\varphi_1 - 2\varphi_{\frac{1}{2}} + \varphi_0)}$$

as required.

Steepest Descent Method Code: `steepest_descent.py`

1.3.2 Newton's Method

A more powerful approach is Newton's method. Instead of using first derivatives, Newton's method uses second-order information from the Hessian matrix:

$$H_g(\mathbf{x}) = \nabla^2 g(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 g}{\partial x_1^2} & \frac{\partial^2 g}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 g}{\partial x_1 \partial x_n} \\ \frac{\partial^2 g}{\partial x_2 x_1} & \frac{\partial^2 g}{\partial x_2^2} & \cdots & \frac{\partial^2 g}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 g}{\partial x_n x_1} & \frac{\partial^2 g}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 g}{\partial x_n^2} \end{bmatrix}$$

It can be derived from applying Newton's method to the root-finding problem $\nabla g(\mathbf{x}) = \mathbf{0}$. The update rule is

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - H_g(\mathbf{x}^{(k-1)})^{-1} \nabla g(\mathbf{x}^{(k-1)})$$

When the starting point is sufficiently close to a minimizer, Newton's method converges quadratically, making it one of the most efficient local methods. However, it requires evaluating and inverting the Hessian, which can be computationally expensive, and may fail if the initial guess is poor, or if the Hessian is not positive definite.

Example 1.3.2.1. Consider again

$$g(x, y) = -4x - 2y - x^2 + 2x^4 + 3y^2$$

Step 0: Start from $\mathbf{x}^{(0)} = (1, 0)^T$. Then the gradient is

$$\nabla g(x, y) = \begin{bmatrix} -4 - 2x + 8x^3 \\ -2 + 6y \end{bmatrix}$$

At $\mathbf{x}^{(0)}$, we have

$$\nabla g(1, 0) = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

Step 1: Computing the Hessian matrix, we have

$$H_g(x, y) = \begin{bmatrix} -2 + 24x^2 & 0 \\ 0 & 6 \end{bmatrix}$$

At $\mathbf{x}^{(0)}$, we have

$$H_g(1, 0) = \begin{bmatrix} 22 & 0 \\ 0 & 6 \end{bmatrix}$$

Step 2: Now we solve

$$H_g(1,0)\mathbf{y} = \nabla g(1,0) \iff \mathbf{y} = \begin{bmatrix} \frac{1}{11} \\ -\frac{1}{3} \end{bmatrix}$$

Step 3: Update

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \mathbf{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{11} \\ -\frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{10}{11} \\ \frac{1}{3} \end{bmatrix}$$

Step 4: (Interpretation) After only one step, Newton's method produces a point already close to the minimizer, reflecting its quadratic convergence when the Hessian is well-conditioned and the initial guess is reasonable.

Remark 1.3.3.

- The **Steepest Descent Method** is derivative-based but simple, requiring only gradients. It guarantees convergence but is often slow.
- The **Newton's Method** exploits curvature through the Hessian, achieving rapid convergence near the solution but at a higher computational cost.

These two methods form the foundation for more advanced algorithms, such as conjugate gradient methods and quasi-Newton schemes, which balance efficiency and practicality.

Newton's Method Code: `newtons_method.py`

Chapter 2

The Preconditioned Conjugate Gradient Methods

In Chapter 1, we studied optimization methods beginning with one-dimensional search techniques (Golden Section and Fibonacci methods), extended to multidimensional search through the Simplex method, and then introduced iterative gradient-based methods such as Steepest Descent and Newton's Method. These approaches revealed the importance of both search directions and step lengths in minimizing objective functions, particularly quadratic functions that naturally arise in solving systems of linear equations.

In this chapter, we focus on a powerful class of iterative methods designed to solve large-scale systems of linear equations, particularly those of the form

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

where A is a symmetric positive definite (SPD) matrix. Direct methods, such as Gaussian elimination or LU decomposition, become computationally expensive for large systems. Instead, iterative methods—especially gradient-based methods—are more efficient. However, basic gradient descent often converges very slowly due to poor conditioning of the matrix A. This motivates the development of the Conjugate Gradient (CG) Method, which accelerates convergence by exploiting conjugacy with respect to the matrix A. Further improvements are achieved through Preconditioning, where the problem is transformed to reduce ill-conditioning, leading to the Preconditioned Conjugate Gradient (PCG) Method.

This chapter develops the theory and practice of these methods in four stages:

We begin by revisiting the gradient method for quadratic minimization problems and introduce the concept of conjugate directions, which generalize orthogonality in optimization. This forms the foundation for more advanced iterative methods.

We present the Conjugate Gradient Method as a direct extension of the conjugate direction framework, showing how it achieves finite-step convergence for SPD systems and is significantly more efficient than steepest descent.

We introduce preconditioning techniques to further enhance convergence rates, especially when dealing with ill-conditioned systems. Several choices of preconditioners will be discussed, together with their impact on practical performance.

Finally, we extend the conjugate gradient idea to nonlinear optimization problems, leading to the family of nonlinear conjugate gradient methods, which play a central role in large-scale optimization.

Through these sections, we will see how the conjugate gradient framework elegantly bridges numerical linear algebra and optimization, providing an essential tool in modern computational mathematics.

2.1 The Gradient Method and The Conjugate Direction Method

In Chapter 1, we introduced the steepest descent method, where the search direction at each step is given by the negative gradient of the objective function. While this method guarantees convergence, its progress can be extremely slow, especially for ill-conditioned problems.

In this section, we first revisit the gradient method in the special case of minimizing quadratic functions of the form

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x},$$

with A symmetric positive definite. We then develop the idea of conjugate directions, a generalization of orthogonal directions, which allows us to construct search directions that avoid repeated minimization along the same dimension. This leads to methods that can reach the exact minimizer in at most n steps for an $n \times n$ system, setting the foundation for the conjugate gradient method introduced in Section 2.2.

2.1.1 The Gradient Method

We consider the quadratic objective function

$$g(\mathbf{x}) = \frac{1}{2}\langle A\mathbf{x}, \mathbf{x}\rangle - \langle \mathbf{b}, \mathbf{x}\rangle + d_0, \quad \mathbf{x} \in \mathbb{R}^n$$

where A is an $n \times n$ symmetric positive definite (SPD) matrix, $\mathbf{b} \in \mathbb{R}^n$, and $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i=1}^n x_i y_i$ is the standard inner product. The minimizer $\mathbf{x}^*$ satisfies

$$A\mathbf{x}^* = \mathbf{b}$$

The search direction for the Gradient method (general form) is given by

$$\mathbf{z}^{(k)} = -\nabla g(\mathbf{x}^{(k-1)}) = \mathbf{b} - A\mathbf{x}^{(k-1)} \equiv \mathbf{r}^{(k-1)}$$

We then update the rule given as

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{z}^{(k)}$$

Two main variants:

1. **Constant Step Length**

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha \mathbf{r}^{(k-1)}, \quad \alpha > 0 \text{ fixed}$$

   Convergence requires $0 < \alpha < \frac{2}{\lambda_{\max}}$, where $\lambda_{\max}$ is the largest eigenvalue of A. Error reduction factor is then given by

$$\gamma = \max_{1 \le i \le n} |1 - \alpha \lambda_i| < 1$$

2. **Optimal Step Length**

$$\alpha_k = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)}\rangle}{\langle A\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)}\rangle} = \frac{\|\mathbf{r}^{(k-1)}\|_2^2}{\langle A\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)}\rangle}$$

   This is equivalent to the Steepest Descent Method for quadratic functions. The error estimate is given by

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_A \le \left(1 - \frac{1}{\rho}\right)^{\frac{k}{2}} \|\mathbf{x}^{(0)} - \mathbf{x}^*\|_A$$

   where $\rho = \frac{\lambda_{\max}}{\lambda_{\min}}$ is the condition number of A and $\|\mathbf{x}\|_A = \sqrt{\langle A\mathbf{x}, \mathbf{x}\rangle}$.

Example 2.1.1.1. We minimize

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T x + 2$$

where

$$A = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

starting from $\mathbf{x}^{(0)} = (0,0)^T$.

For a quadratic, $\nabla g(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, so the residual

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} = -\nabla g(\mathbf{x}^{(k)})$$

The gradient method with residual direction uses

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{r}^{(k)}$$

Method I (Constant step length $\alpha = 0.2$): We have

$$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

So then

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + 0.2\mathbf{r}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.2\begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix}$$

Compute the new residual

$$A\mathbf{x}^{(1)} = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix}\begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 0 \\ 3.4 \end{bmatrix}$$

and so

$$\mathbf{r}^{(1)} = \mathbf{b} - A\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ -0.4 \end{bmatrix}$$

(Optional objective check)

$$g(\mathbf{x}^{(1)}) = \frac{1}{2}\mathbf{x}^{(1)T} A\mathbf{x}^{(1)} - \mathbf{b}^T\mathbf{x}^{(1)} + 2$$

Since $A\mathbf{x}^{(1)} = (0, 3.4)^T$, we get $\mathbf{x}^{(1)T}A\mathbf{x}^{(1)} = (0.2, 0.6)\cdot(0, 3.4) = 2.04$. Therefore,

$$g(\mathbf{x}^{(1)}) = \frac{1}{2}(2.04) - 2 + 2 = 1.02$$

Then we find $\mathbf{x}^{(2)}$, in which

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + 0.2\mathbf{r}^{(1)} = \begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix} + 0.2 \begin{bmatrix} 1 \\ -0.4 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.52 \end{bmatrix}$$

The residual and (optional) value is given by

$$A\mathbf{x}^{(2)} = \begin{bmatrix} 0.68 \\ 2.72 \end{bmatrix}, \quad \mathbf{r}^{(2)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 0.68 \\ 2.72 \end{bmatrix} = \begin{bmatrix} 0.32 \\ 0.28 \end{bmatrix}$$

Then

$$g(\mathbf{x}^{(2)}) = \frac{1}{2}(0.4, 0.52) \cdot (0.68, 0.72) - (0.4 + 1.56) + 2 = 0.883$$

Method II (Optimal step length (exact 1D line search along $\mathbf{r}^{(k)}$): For a quadratic with search direction $\mathbf{p}^{(k)} = \mathbf{r}^{(k)}$

$$\alpha_k = \arg\min_{\alpha \geq 0} g(\mathbf{x}^{(k)} + \alpha \mathbf{r}^{(k)}) = \frac{\mathbf{r}^{(k)T}\mathbf{r}^{(k)}}{\mathbf{r}^{(k)T} A \mathbf{r}^{(k)}}$$

For the first step, we have

$$\mathbf{r}^{(0)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad A\mathbf{r}^{(0)} = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 17 \end{bmatrix}$$

So then,

$$\alpha_0 = \frac{\|\mathbf{r}^{(0)}\|_2^2}{\mathbf{r}^{(0)T} A \mathbf{r}^{(0)}} = \frac{10}{51}$$

and so,

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0 \mathbf{r}^{(0)} = \frac{10}{51} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{10}{51} \\ \frac{30}{51} \end{bmatrix}$$

Update residual exactly,

$$A\mathbf{x}^{(1)} = \begin{bmatrix} 0 \\ \frac{170}{51} \end{bmatrix}, \quad \mathbf{r}^{(1)} = \mathbf{b} - A\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ -\frac{1}{3} \end{bmatrix}$$

(Nice property: in steepest descent for quadratics, successive residuals are Euclidean orthogonal: $\mathbf{r}^{(1)} \perp \mathbf{r}^{(0)}$. Indeed, $1 \cdot 1 + 3 \cdot \left(-\frac{1}{3}\right) = 0$.

For the second step, we do a similar computation:

$$A\mathbf{r}^{(1)} = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ -\frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{10}{3} \\ -3 \end{bmatrix},$$

$$\alpha_1 = \frac{\|\mathbf{r}^{(1)}\|_2^2}{\mathbf{r}^{(1)T} A \mathbf{r}^{(1)}} = \frac{10}{39}$$

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha_1 \mathbf{r}^{(1)} = \begin{bmatrix} \frac{100}{221} \\ \frac{1000}{1989} \end{bmatrix}$$

Theorem 2.1.1.2. *Let A be an $n \times n$ symmetric positive definite matrix with eigenvalues*

$$0 < \lambda_{\min} \le \lambda_i \le \lambda_{\max}, \quad i = 1, 2, ..., n.$$

Suppose $0 < \alpha < \frac{2}{\lambda_{\max}}$. Then the gradient method with constant step length converges for any initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and the error satisfies

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\| \le \gamma^k \|\mathbf{x}^{(0)} - \mathbf{x}^*\|, \quad \ge 1,$$

where

$$\gamma = \max_{1 \le i \le n} |1 - \alpha\lambda_i| < 1$$

and $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$ is the Euclidean 2-norm.

Proof. Recall the gradient iteration

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha(\mathbf{b} - A\mathbf{x}^{(k-1)})$$

Let $\mathbf{x}^*$ be the exact solution of $A\mathbf{x}^* = \mathbf{b}$. Then

$$\mathbf{x}^* = \mathbf{x}^* + \alpha(\mathbf{b} - A\mathbf{x}^*)$$

Subtracting these,

$$\mathbf{e}^{(k)} := \mathbf{x}^*(k) - \mathbf{x}^* = (I - \alpha A)\mathbf{e}^{(k-1)}$$

By induction,

$$\mathbf{e}^{(k)} = (I - \alpha A)^k \mathbf{e}^{(0)}$$

Since A is SPD, it has an orthonormal eigenbasis $\{\mathbf{v}_i\}_{i=1}^n$ with eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. Writing $\mathbf{e}^{(0)} = \sum_{i=1}^n c_i \mathbf{v}_i$, we get

$$\mathbf{e}^{(k)} = \sum_{i=1}^n c_i (1 - \alpha\lambda_i)^k \mathbf{v}_i$$

Thus,

$$\|\mathbf{e}^{(k)}\| \le \gamma^k \|\mathbf{e}^{(0)}\|, \quad \text{where } \gamma = \max_{1 \le i \le n} |1 - \alpha\lambda_i|$$

For convergence, we need $\gamma < 1$, that is, for all i,

$$|1 - \alpha\lambda_i| < 1$$

Since $\lambda_i > 0$ and $\alpha > 0$, this inequality is equivalent to

$$0 < \alpha\lambda_i < 2$$

for all i, which requires $\alpha\lambda_{\max} < 2$. Therefore,

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

$\square$

Remark 2.1.1.3. We need to use $0 < \alpha < \frac{2}{\lambda_{\max}}$ for the Gradient method with constant step length. For example, choose $\alpha = \frac{1}{\lambda_{\max}}$. Then

$$\gamma = \max_{1\le i\le n} \left|1 - \frac{\lambda_i}{\lambda_{\max}}\right| = 1 - \frac{\lambda_{\min}}{\lambda_{\max}} = 1 - \frac{1}{\rho} < 1$$

where $\rho = \frac{\lambda_{\max}}{\lambda_{\min}} > 0$ which is called the conditional number of matrix A.

- If $\rho \approx 1$ (good matrix A), then $\gamma \approx 0$, it converges fast.
- If $\rho \gg 1$ (bad matrix A), then $\gamma \approx 1$, it converges very slowly.

Gradient Method Code: `gradient_method.py`

2.1.2 The Conjugate Direction Method

In the gradient descent (steepest descent), search directions are chosen as residuals $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$. Although convergence is guaranteed, the directions are not independent, leading to repeated progress in the same subspace and slow "zig-zag" behaviour when A is ill-conditioned.

To overcome this, we introduce the concept of conjugate directions.

Definition 2.1.2.1. For a symmetric positive definite matrix A, a set of nonzero vectors $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\} \subseteq \mathbb{R}^n$ is called A-conjugate if for all $i \neq j$,

$$\langle \mathbf{u}_i, A\mathbf{u}_j \rangle = 0$$

and for all i

$$\langle \mathbf{u}_i, A\mathbf{u}_i \rangle > 0$$

This generalizes orthogonality: in Euclidean space, orthogonal vectors satisfy $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = 0$. For a quadratic minimization, A-conjugacy ensures that minimizing along one direction does not spoil minimization along previous ones.

Algorithm (Conjugate Direction Method): Let $\{\mathbf{u}_i\}_{i=1}^n$ be an A-conjugate set. Then

1. Choose $\mathbf{x}^{(0)} \in \Omega$.
2. For $k = 1, 2, ..., n$,
    - Compute step size
    $$t_k = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{u}_k \rangle}{\langle A\mathbf{u}_k, \mathbf{u}_k \rangle}, \quad \mathbf{r}^{(k-1)} = \mathbf{b} - A\mathbf{x}^{(k-1)}$$
    - Update solution
    $$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + t_k \mathbf{u}_k$$

    Here, t_k is the optimal step length along u_k.

Theorem 2.1.2.2. *If $\{\mathbf{u}_i\}_{i=1}^n$ is an A-conjugate set, then after n steps, the method produces the exact solution:*

$$\mathbf{x}^{(n)} = \mathbf{x}^*$$

where $A\mathbf{x}^* = \mathbf{b}$.

Proof Sketch. Since $\mathbf{x}^{(0)}$ and the n directions span $\mathbb{R}^n$, the updates explore the full space without redundancy. A-conjugacy ensures independence, so minimization terminates exactly at $\mathbf{x}^*$. □

Example 2.1.2.3. In $\mathbb{R}^2$, suppose $\mathbf{u}_1$ and $\mathbf{u}_2$ are chosen to be A-conjugate. After two steps, the exact minimizer $\mathbf{x}^*$ is reached.

- In contrast, steepest descent may require many steps, since successive gradients are only orthogonal (in the Euclidean sense), not A-conjugate.
- Geometrically: conjugate directions (red) go straight to the minimizer, while steepest descent (green) zig-zags across surface level curves.

Example 2.1.2.4. We solve the quadratic $g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, where

$$A = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The conjugate direction method assumes we have (or construct) A-conjugate directions $\{\mathbf{u}_1, \mathbf{u}_2\}$ such that

$$\langle A\mathbf{u}_i, \mathbf{u}_j \rangle = 0$$

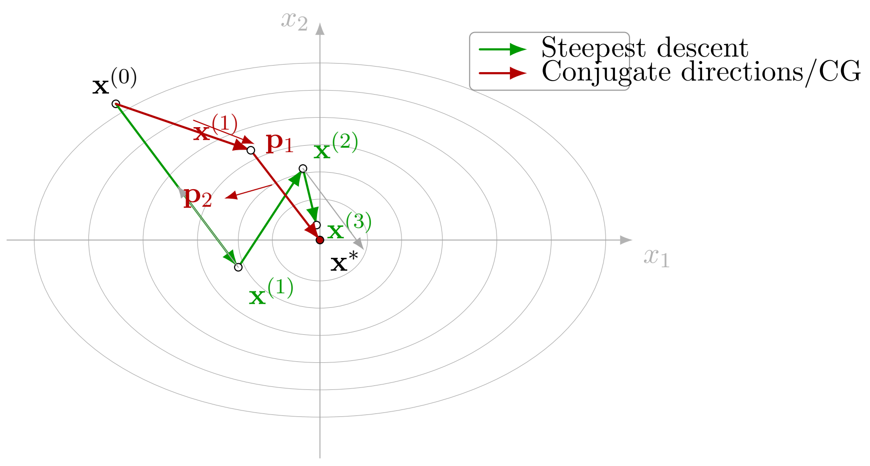


Figure 2.1: Quadratic level sets (ellipses) with steepest descent (green, zig-zag) versus conjugate directions/CG (red, two steps) approaching the minimizer $\mathbf{x}^*$.

for all $i \neq j$, and

$$\langle A\mathbf{u}_i, \mathbf{u}_i \rangle > 0$$

for all i.

At the kth step, we minimize along $\mathbf{u}_k$,

$$t_k = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{u}_k \rangle}{\langle A\mathbf{u}_k, \mathbf{u}_k \rangle}, \quad \mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + t_k \mathbf{u}_k, \quad \mathbf{r}^{(k-1)} = \mathbf{b} - A\mathbf{x}^{(k-1)}$$

Step 0 (Build an A-conjugate pair $\{\mathbf{u}_1, \mathbf{u}_2\}$): A handy choice is $\mathbf{u}_1 = \mathbf{r}^{(0)}$. With $\mathbf{x}^{(0)} = (0, 0)$, we have

$$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{u}_1 = \mathbf{r}^{(0)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Now we need to find $\mathbf{u}_2$ such that $\langle A\mathbf{u}_2, \mathbf{u}_1 \rangle = 0$.

Indeed, let $\mathbf{v}_2 = \mathbf{e}_1 = (1, 0)^T$ (i.e. the standard basis vector). Then

$$A\mathbf{v}_2 = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

So then

$$\langle A\mathbf{v}_2, \mathbf{u}_1 \rangle = (3, -1) \cdot (1, 3) = 3 - 3 = 0$$

So $\mathbf{u}_2 = \mathbf{e}_1$ is already A-conjugate to $\mathbf{u}_1$. (There is another way that we can construct a second vector via the Gram–Schmidt Orthogonalization Process; see the remark at the end).

Step 1 (Line search along $\mathbf{u}_1$): Compute the denominator of $\langle A\mathbf{u}_1, \mathbf{u}_1\rangle$. Indeed,

$$A\mathbf{u}_1 = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix}\begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 17 \end{bmatrix}, \quad \langle A\mathbf{u}_1, \mathbf{u}_1\rangle = (0, 17)\cdot(1, 3) = 51$$

Numerator $\langle \mathbf{r}^{(0)}, \mathbf{u}_1\rangle = \|\mathbf{r}^{(0)}\|_2^2 = 1^2 + 3^2 = 10$. Then,

$$t_1 = \frac{\langle \mathbf{r}^{(0)}, \mathbf{u}_1\rangle}{\langle A\mathbf{u}_1, \mathbf{u}_1\rangle} = \frac{10}{51}$$

Then we update solution

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + t_1\mathbf{u}_1 = \frac{10}{51}\begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{10}{51} \\ \frac{30}{51} \end{bmatrix}$$

Residual update

$$\mathbf{r}^{(1)} = \mathbf{b} - A\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \frac{10}{51}\begin{bmatrix} 0 \\ 17 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{17}{51} \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{1}{3} \end{bmatrix}$$

Step 2 (Line search along $\mathbf{u}_2$): Compute the denominator $\langle A\mathbf{u}_2, \mathbf{u}_2\rangle$:

$$A\mathbf{u}_2 = A\mathbf{e}_1 = \begin{bmatrix} 3 \\ -1 \end{bmatrix}, \quad \langle A\mathbf{u}_2, \mathbf{u}_2\rangle = (3, -1)\cdot(1, 0) = 3$$

The numerator is $\langle \mathbf{r}^{(1)}, \mathbf{u}_2\rangle = \mathbf{r}^{(1)}\cdot\mathbf{e}_1 = 1$. Then,

$$t_2 = \frac{\langle \mathbf{r}^{(1)}, \mathbf{u}_2\rangle}{\langle A\mathbf{u}_2, \mathbf{u}_2\rangle} = \frac{1}{3}$$

Then we update solution

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + t_2\mathbf{u}_2 = \begin{bmatrix} \frac{10}{51} \\ \frac{30}{51} \end{bmatrix} + \frac{1}{3}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{9}{17} \\ \frac{10}{17} \end{bmatrix}$$

This is exactly the solution of $A\mathbf{x} = \mathbf{b}$ (which can be verified by solving the 2×2 system). Hence, the method terminates in two steps.

Since $\mathbf{u}_1$ and $\mathbf{u}_2$ are A-conjugate, minimizing along $\mathbf{u}_2$ does not undo the minimization along $\mathbf{u}_1$. In $\mathbb{R}^2$, two such steps reach the unique minimizer.

Remark 2.1.2.5. As noted in Step 0 of the above example, we can apply the Gram-Schmidt Orthogonalization Process to construct $\mathbf{u}_2$. Indeed, if we had started with $\mathbf{v}_2 = \mathbf{e}_2 = (0,1)^T$, we would orthogonalize in the A-inner product

$$\mathbf{u}_2 = \mathbf{v}_2 - \frac{\langle A\mathbf{v}_2, \mathbf{u}_1\rangle}{\langle A\mathbf{u}_1, \mathbf{u}_1\rangle}\mathbf{u}_1, \quad A\mathbf{v}_2 = \begin{bmatrix} -1 \\ 6 \end{bmatrix}, \quad \langle A\mathbf{v}_2, \mathbf{u}_1\rangle = 17$$

Since $\langle A\mathbf{u}_1, \mathbf{u}_1\rangle = 51$, we get $\mathbf{u}_2 = \mathbf{v}_2 - \frac{17}{51} = \mathbf{u}_1 = \begin{bmatrix} -\frac{1}{3} \\ 0 \end{bmatrix}$, which is just a negative multiple of $\mathbf{e}_1$, still perfectly valid and A-conjugate to $\mathbf{u}_1$.

The Conjugate Gradient Method (Section 2.2) is a practical implementation of this idea. Instead of precomputing an entire A-conjugate basis, CG constructs conjugate directions dynamically using residuals and short recursions.

2.2 The Conjugate Gradient Method (CG)

The gradient method with optimal step length (steepest descent) improves upon constant-step descent but still converges slowly when the system matrix A is ill-conditioned. The conjugate direction method achieves finite-step convergence by using a set of A-conjugate directions, but it requires the entire set of directions in advance.

The Conjugate Gradient (CG) Method combines both ideas. It generates A-conjugate directions dynamically from residuals, allowing fast convergence without precomputing a conjugate basis. CG is especially effective for large sparse SPD systems.

In the conjugate direction method, directions $\mathbf{u}_k$ are A-conjugate:

$$\langle A\mathbf{u}_i, \mathbf{u}_j\rangle = 0 \quad \forall i \neq j$$

If such a set of n conjugate directions is available, the solution is obtained in at most n steps. The CG method constructs these directions iteratively, using only the current residual.

Algorithm

1. Choose $\mathbf{x}^{(0)} \in \Omega$. Set residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{x}^{(0)}$, and initial search direction $\mathbf{z}^{(1)} = \mathbf{r}_0$.

2. For $k \geq 1$:

- Step length:

$$\alpha_k = \frac{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle}{\langle A\mathbf{z}^{(k)}, \mathbf{z}^{(k)} \rangle} = \frac{\|\mathbf{r}^{(k-1)}\|_2^2}{\langle A\mathbf{z}^{(k)}, \mathbf{z}^{(k)} \rangle}$$

- Update solution:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{z}^{(k)}$$

- Update residual:

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \alpha_k A\mathbf{z}^{(k)}$$

- Compute direction coefficient:

$$\beta_k = \frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle}{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle} = \frac{\|\mathbf{r}^{(k)}\|_2^2}{\|\mathbf{r}^{(k-1)}\|_2^2}$$

- Update direction:

$$\mathbf{z}^{(k+1)} = \mathbf{r}^{(k)} + \beta_k \mathbf{z}^{(k)}$$

Proposition 2.2.1. *1. **Conjugacy:** $\langle A\mathbf{z}^{(i)}, \mathbf{z}^{(j)} \rangle = 0$ for all $i \neq j$.*

*2. **Residual orthogonality:** $\langle \mathbf{r}^{(i)}, \mathbf{r}^{(j)} \rangle = 0$ for all $i \neq j$.*

*3. **Finite termination:** Exact solution in at most n steps.*

Definition 2.2.2. Let $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$ be the condition number of A. Then the error estimate is given as

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|\mathbf{x}^{(0)} - \mathbf{x}^*\|_A$$

Remark 2.2.3. CG finds the exact solution in at most n steps, but in practice for large n, a very good approximation is often achieved in $\mathcal{O}(\sqrt{n})$ steps. It is particularly efficient for large, sparse, SPD linear systems. Compared to steepest descent, CG avoids the zig-zag behavior and exploits conjugacy to accelerate convergence.

Example 2.2.4. Consider $g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ (equivalently solve $A\mathbf{x} = \mathbf{b}$) with

$$A = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Step 1: Note that $\mathbf{r}_0 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$ and $\mathbf{z}^{(1)} = \mathbf{r}_0$, so then

$$A\mathbf{z}^{(1)} = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 17 \end{bmatrix}$$

Computing the step length, we have

$$\alpha_1 = \frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{z}^{(1)T} A \mathbf{z}^{(1)}} = \frac{10}{51}$$

and thus, updating solution

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_1 \mathbf{z}^{(1)} = \frac{10}{51} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{10}{51} \\ \frac{30}{51} \end{bmatrix}$$

Updating the residual gives

$$\mathbf{r}_1 = \mathbf{r}_0 - \alpha_1 A\mathbf{z}^{(1)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \frac{10}{51} \begin{bmatrix} 0 \\ 17 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{1}{3} \end{bmatrix}$$

Computing the direction coefficient, we have

$$\beta_1 = \frac{\mathbf{r}_1^T \mathbf{r}_1}{\mathbf{r}_0^T \mathbf{r}_0} = \frac{1}{9}$$

and update direction

$$\mathbf{z}^{(2)} = \mathbf{r}_1 + \beta_1 \mathbf{z}^{(1)} = \begin{bmatrix} \frac{10}{9} \\ 0 \end{bmatrix}$$

Step 2: Note that

$$A\mathbf{z}^{(2)} = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} \frac{10}{9} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{10}{3} \\ -\frac{10}{9} \end{bmatrix}$$

Then computing the step length, we have

$$\alpha_2 = \frac{\mathbf{r}_1^T \mathbf{r}_1}{\mathbf{z}^{(2)T} A \mathbf{z}^{(2)}} = \frac{27}{90} = \frac{3}{10}$$

and thus, updating solution

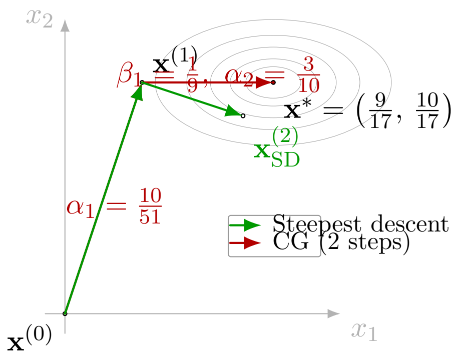
$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha_2 \mathbf{z}^{(2)} = \begin{bmatrix} \frac{9}{17} \\ \frac{10}{17} \end{bmatrix}$$

Updating the residual gives

$$\mathbf{r}_2 = \mathbf{r}_1 - \alpha_2 A\mathbf{z}^{(2)} = \mathbf{0} \implies \mathbf{x}^{(2)} = \mathbf{x}^*$$

Therefore, the CG converges in at most $n = 2$ steps here, yielding the exact solution

$$\mathbf{x}^* = \left(\frac{9}{17}, \frac{10}{17}\right)^T$$



x_2
x_1
$\mathbf{x}^{(0)}$
$\mathbf{x}^{(1)}$
$\beta_1 = \frac{1}{9}$,
$\alpha_2 = \frac{3}{10}$
$\mathbf{x}^* = \left(\frac{9}{17}, \frac{10}{17}\right)$
$\mathbf{x}^{(2)}_{\mathrm{SD}}$
$\alpha_1 = \frac{10}{51}$
Steepest descent
CG (2 steps)

Figure 2.2: CG (red) versus steepest descent (green) on the quadratic with $A = \left(\begin{smallmatrix} 3 & -1 \\ -1 & 6 \end{smallmatrix}\right)$, $b = (1, 3)^\top$. Both share the first step to $\mathbf{x}^{(1)}$, then CG reaches the exact minimizer $\mathbf{x}^*$ in the second step, while steepest descent continues to zig-zag.

CG Method Code: `cg_method.py`

2.3 The Preconditioned Conjugate Gradient Method (PCG)

The Conjugate Gradient (CG) method provides a powerful iterative framework for solving symmetric positive definite (SPD) systems $A\mathbf{x} = \mathbf{b}$ by constructing A-conjugate search directions dynamically from residuals. While CG converges rapidly for well-conditioned systems, its performance can deteriorate severely when the condition number $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$ is large. In such cases, the error bound

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_A \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k \|\mathbf{x}^{(0)} - \mathbf{x}^*\|_A$$

shows that convergence becomes very slow if $\kappa(A) \gg 1$.

The Preconditioned Conjugate Gradient Method (PCG) addresses this challenge by applying a preconditioner C^{-1} to transform the original system into an equivalent one with more favourable spectral properties. Specifically, we consider the transformation $\mathbf{y} = C^T\mathbf{x}$ which leads to a new quadratic minimization problem involving the modified matrix

$$A^* = C^{-1}AC^{-T}$$

If C is chosen so that A^* has a smaller condition number, then CG applied to this transformed system converges significantly faster. The resulting iterates are mapped to approximations of the solution to the original system.

In practice, PCG avoids forming A^* explicitly. Instead, it incorporates C^{-1} directly into the CG iteration by introducing preconditioned residuals and corresponding search directions. This allows the algorithm to retain the efficiency and short recurrence structure of CG, while benefiting from accelerated convergence.

The design of a good preconditioner C^{-1} is problem dependent. Common choices include:

1. Diagonal scaling $C^{-1} = D^{-\frac{1}{2}}$, where D is the diagonal matrix of A.
2. Incomplete factorizations such as $C^{-1} = L^{-1}$ if $A \approx LL^T$.
3. Block preconditioners for block-structured systems.

In this section, we develop the PCG algorithm, derive its iteration formulas, and discuss strategies for constructing effective preconditioners. We also highlight its theoretical properties and illustrate how PCG greatly outperforms standard CG in solving ill-conditioned SPD systems.

Consider

$$g(\mathbf{x}) = \frac{1}{2}\langle \mathbf{x}, A\mathbf{x}\rangle - \langle \mathbf{b}, \mathbf{x}\rangle + d_0$$

where A is a symmetric positive definite matrix. Introduce a singular matrix C and let

$$\mathbf{y} = C^T\mathbf{x}, \quad \mathbf{x} = C^{-T}\mathbf{y}$$

Define the transform quadratic

$$G(\mathbf{y}) = g(C^{-1}\mathbf{y}) = \frac{1}{2}\langle \mathbf{y}, A^*\mathbf{y}\rangle - \langle \mathbf{b}^*, \mathbf{y}\rangle + d_0$$

with

$$A^* = C^{-1}AC^{-T}, \quad \mathbf{b}^* = C^{-1}\mathbf{b}$$

If A^* has a smaller condition number than A, then CG on $A^*\mathbf{y} = \mathbf{b}^*$ converges much faster. The matrix C^{-1} is called the preconditioned matrix.

We avoid forming A^* directly. Instead, PCG introduces preconditioned residuals.

1. Start with $\mathbf{x}^{(0)} \in \Omega$.
   - Residual: $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$.
   - Preconditioned residual: $\mathbf{w}^{(0)} = C^{-1}\mathbf{r}^{(0)}$.
   - Direction: $\mathbf{z}^{(1)} = C^{-1}\mathbf{w}^{(0)}$.
2. For $k \geq 1$,
   - Step length:
   $$\alpha_k = \frac{\langle \mathbf{w}_{k-1}, \mathbf{w}_{k-1} \rangle}{\langle A\mathbf{z}^{(k)}, \mathbf{z}^{(k)} \rangle}$$
   - Update iterate:
   $$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{z}^{(k)}$$
   - Update residual:
   $$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \alpha_k A\mathbf{z}^{(k)}$$
   - Preconditioned residual:
   $$\mathbf{w}_k = C^{-1}\mathbf{r}^{(k)}$$
   - Direction coefficient:
   $$\beta_k = \frac{\langle \mathbf{w}_k, \mathbf{w}_k \rangle}{\langle \mathbf{w}_{k-1}, \mathbf{w}_{k-1} \rangle}$$
   - Update direction
   $$\mathbf{z}^{(k+1)} = C^{-1}\mathbf{w}_k + \beta_k \mathbf{z}^{(k)}$$

Note that if $C^{-1} = I$, PCG reduces to standard CG.

There are three choices of preconditioners:

1. Diagonal scaling

$$D^{\frac{1}{2}} = \text{diag}(\sqrt{a_{ii}}), \quad C^{-1} = D^{-\frac{1}{2}}$$

2. Cholesky-based: If $A = LL^T$, take $C^{-1} = L^{-1}$. Then
$$A^* = C^{-1}AC^{-T} = I$$
giving convergence in one step.

3. Block preconditioners: For block matrices $A = [A_{ij}]$, choose C^{-1} block diagonal with L_j^{-1} from factorizations $A_{ii} = L_i L_i^T$.

Example 2.3.1. We solve $A\mathbf{x} = \mathbf{b}$ (equivalently, minimize $g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$) with
$$A = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
Use the diagonal preconditioner
$$D = \operatorname{diag}(A) = \begin{bmatrix} 3 & 0 \\ 0 & 6 \end{bmatrix}, \quad C^{-1} = D^{-\frac{1}{2}} = \operatorname{diag}\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{6}}\right)$$
(Equivalently, $D^{-1} = \operatorname{diag}\left(\frac{1}{3}, \frac{1}{6}\right)$.

Step 1: We first compute the residual and preconditioned objects:
$$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{w}^{(0)} = C^{-1}\mathbf{r}^{(0)} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{3}{\sqrt{6}} \end{bmatrix}$$
Then we compute the initial search direction given by
$$\mathbf{z}^{(1)} = C^{-T}\mathbf{w}^{(0)} = C^{-1}\mathbf{w}^{(0)} = D^{-1}\mathbf{r}^{(0)} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{2} \end{bmatrix}$$
Then compute the step
$$A\mathbf{z}^{(1)} = \begin{bmatrix} \frac{1}{2} \\ \frac{8}{3} \end{bmatrix}, \quad \langle \mathbf{w}^{(0)}, \mathbf{w}^{(0)} \rangle = \mathbf{r}^{(0)T} D^{-1}\mathbf{r}^{(0)} = \frac{11}{6}, \quad \langle A\mathbf{z}^{(1)}, \mathbf{z}_1 \rangle = \frac{3}{2}$$
Thus,
$$\alpha_1 = \frac{\langle \mathbf{w}^{(0)}, \mathbf{w}^{(0)} \rangle}{\langle A\mathbf{z}^{(1)}, \mathbf{z}^{(1)} \rangle} = \frac{\frac{11}{6}}{\frac{3}{2}} = \frac{11}{9} = 1.2222$$
Now we update
$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_1 \mathbf{z}^{(1)} = \frac{11}{9}\begin{bmatrix} \frac{1}{3} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{11}{27} \\ \frac{11}{18} \end{bmatrix}$$
and update the residual
$$\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha_1 A\mathbf{z}^{(1)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \frac{11}{9}\begin{bmatrix} \frac{1}{2} \\ \frac{8}{3} \end{bmatrix} = \begin{bmatrix} \frac{7}{18} \\ -\frac{7}{27} \end{bmatrix}$$

Step 2: We compute the preconditioned residual and direction update:

$$\mathbf{w}^{(1)} = C^{-1}\mathbf{r}^{(1)}, \quad \langle \mathbf{w}^{(1)}, \mathbf{w}^{(1)} \rangle = \mathbf{r}^{(1)T} D^{-1} \mathbf{r}^{(1)} = \frac{539}{8748}$$

and

$$\beta_1 = \frac{\langle \mathbf{w}_1, \mathbf{w}_1 \rangle}{\langle \mathbf{w}_0, \mathbf{w}_0 \rangle} = \frac{539/8748}{11/6} = \frac{49}{1458}$$

The Raw (Cholesky) direction from $\mathbf{w}^{(1)}$ is given by

$$C^{-T}\mathbf{w}^{(1)} = D^{-1}\mathbf{r}^{(1)} = \begin{bmatrix} \frac{7}{54}, & -\frac{7}{162} \end{bmatrix}^T$$

The PCG direction $\mathbf{z}^{(2)}$ is given by

$$\mathbf{z}^{(2)} = C^{-T}\mathbf{w}^{(1)} + \beta_1 \mathbf{z}^{(1)} = \begin{bmatrix} \frac{7}{54} \\ -\frac{7}{162} \end{bmatrix} + \frac{49}{1458} \begin{bmatrix} \frac{1}{3} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{308}{2187} \\ -\frac{77}{2916} \end{bmatrix}$$

Computing the step, we have

$$A\mathbf{z}^{(2)} = \begin{bmatrix} 3 & -1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} \frac{308}{2187} \\ -\frac{77}{2916} \end{bmatrix}$$

and thus,

$$\alpha_2 = \frac{\langle \mathbf{w}^{(1)}, \mathbf{w}^{(1)} \rangle}{\langle A\mathbf{z}^{(2)}, \mathbf{z}^{(2)} \rangle} = \frac{162}{187}$$

Finally, we update

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha_2 \mathbf{z}^{(2)} = \begin{bmatrix} \frac{9}{17} \\ \frac{10}{17} \end{bmatrix}$$

and $\mathbf{r}^{(2)} = \mathbf{0}$.

Conclusion: PCG reaches the exact solution in two steps and thus,

$$\mathbf{x}^* = \begin{bmatrix} \frac{9}{17}, & \frac{10}{17} \end{bmatrix}^T$$

Remark 2.3.2. On $\mathbb{R}^n$, both CG and PCG terminate in at most n steps, so you will not see iteration count savings in two dimensions.

Remark 2.3.3. The mechanics differ: PCG uses preconditioned residuals $\mathbf{w}^{(k)} = C^{-1}\mathbf{r}^{(k)}$ and directions $\mathbf{z}^{(k)} = C^{-T}\mathbf{w}^{(k)} + \beta_k \mathbf{z}^{(k-1)}$ and its step sizes α_k and coefficients β_k are built from $\langle \mathbf{w}^{(k)}, \mathbf{w}^{(k)} \rangle$ inner products.

Remark 2.3.4. In higher dimensions and with ill-conditioned A, a good preconditioner can dramatically reduce the number of iterations compared to plain CG by shrinking the effective condition number of the transformed matrix $A^* = C^{-1}AC^{-T}$.

PCG Method Code: `pcg_method.py`

2.4 The Nonlinear CG Methods

The Conjugate Gradient method was originally developed for quadratic minimization problems, where the objective function takes the form

$$g(\mathbf{x}) = \frac{1}{2}\langle \mathbf{x}, A\mathbf{x}\rangle - \langle \mathbf{b}, \mathbf{x}\rangle$$

where A is a symmetric positive definite matrix. In this case, the method terminates in at most n steps and enjoys strong theoretical guarantees.

However, in practice, we often face general nonlinear optimization problems

$$g(\mathbf{x}^*) = \min_{\mathbf{x}\in\Omega} g(\mathbf{x}), \quad \Omega \subseteq \mathbb{R}^n$$

where $g(\mathbf{x})$ is not quadratic, and $\mathbf{x}^*$ is a local minimizer. For such problems, steepest descent remains inefficient, and standard CG cannot be applied directly because its derivation relies on linearity and A-conjugacy.

The Nonlinear Conjugate Gradient (NCG) method extend the ideas of CG to this broader class of problems. The key ingredients are:

- Search direction: updated as a combination of the negative gradient and the previous search direction

  $$\mathbf{z}^{(k+1)} = -\nabla g(\mathbf{x}^{(k)}) + \beta_k \mathbf{z}^{(k)}$$

  with different formulas for β_k depending on the variant.

- Line search: at each step, the step length α_k is chosen to approximately minimize $g(\mathbf{x}^{(k-1)} + \alpha \mathbf{z}^{(k)})$.

- Variants: several choices of β_k exist, such as the
  - Fletcher–Reeves (FR) method,
  - Polak–Ribière (PR) method,

  which reduce to the standard CG formulas when g is quadratic.

Nonlinear CG methods thus inherit much of the efficiency of linear CG while being applicable to large-scale unconstrained nonlinear optimization problems. They play a central role in scientific computing, machine learning, and applied mathematics where memory-efficient iterative methods are essential.

In this section, we introduce the nonlinear CG framework, describe the FR and PR variants, and discuss practical issues such as line search strategies and convergence behaviour.

We want to solve

$$g(\mathbf{x}^*) = \min_{\mathbf{x}\in\Omega}, \quad \Omega \subseteq \mathbb{R}^n$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a smooth (differentiable) function and $\mathbf{x}^*$ is a local minimizer. The gradient condition at the minimizer is

$$\nabla g(\mathbf{x}^*) = 0.$$

The general framework of the line search method is given by

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{z}^{(k)}$$

where

- $\mathbf{z}^{(k)}$ is the search direction,
- α_k is the step length chosen by a line search.

We require

$$\alpha_k \approx \frac{1}{2}\left(0.5 - \frac{h_1^*}{h_3^*}\right)$$

where h_1^* and h_3^* come from the interpolation of the objective function along the line.

Algorithm of Nonlinear Conjugate Gradient Method:

1. Initialization:
   - Choose initial guess $\mathbf{x}^{(0)} \in \Omega$.
   - Set initial search direction

$$\mathbf{z}^{(1)} = -\nabla g(\mathbf{x}^{(0)})$$

2. For $k \geq 1$:
   - Compute α_k by approximate line search.
   - Update iterate

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{z}^{(k)}$$

- Update search direction

$$\mathbf{z}^{(k+1)} = -\nabla g(\mathbf{x}^{(k)}) + \beta_k \mathbf{z}^{(k)}$$

Different NCG variants are defined by the choice of β_k.

1. Fletcher–Reeves (FR):

$$\beta_k^{\mathrm{FR}} = \frac{\|\nabla g(\mathbf{x}^{(k)})\|^2}{\|\nabla g(\mathbf{x}^{(k-1)})\|^2}.$$

   - Direct extension of linear CG formula $\frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle}{\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle}$.
   - Ensures global convergence under exact line search.

2. Polak–Ribière (PR)

$$\beta_k^{\mathrm{PR}} = \frac{\nabla g(\mathbf{x}^{(k)})^T (\nabla g(\mathbf{x}^{(k)}) - \nabla g(\mathbf{x}^{(k-1)}))}{\|\nabla g(\mathbf{x}^{(k-1)})\|^2}$$

   - Reduces to FR when g is quadratic (since gradients are orthogonal).
   - Often performs better in practice.
   - To avoid instability, many implementations use $\beta_k = \max\{0, \beta_k^{\mathrm{PR}}\}$.

Proposition 2.4.1 (**Descent Property**)**.** *If α_k satisfies Wolfe conditions and β_k is chosen as FR or PR, then $\mathbf{z}^{(k)}$ is a descent direction.*

Proposition 2.4.2 (**Global Convergence**)**.** *Under mild assumptions, FR and PR methods converge globally.*

Remark 2.4.3. Like CG, NCG requires only gradient evaluations and vector operations, so no Hessian is required.

Remark 2.4.4. In terms of practical performance, PR is often faster, but less stable. On the other hand, FR is more stable, but sometimes slower.

Example 2.4.5. Consider the nonlinear function

$$g(x, y) = (x-1)^2 + (y-2)^2 + \sin(xy)$$

We apply Nonlinear CG with Fletcher (FR) and Polak–Ribière (PR) variants.

Step 0: Start at $\mathbf{x}^{(0)} = \mathbf{0}$. The gradient is given as

$$\nabla g(x, y) = \begin{bmatrix} 2(x-1) + y\cos(xy) \\ 2(y-2) + x\cos(xy) \end{bmatrix}$$

and at $\mathbf{x}^{(0)}$, we have $\nabla g = (-2, -4)^T$. The initial direction is then given as

$$\mathbf{z}^{(1)} = -\nabla g(\mathbf{x}^{(0)}) = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Step 1: We will approximate α_1 by quadratic interpolation. Define $\varphi(\alpha) = g(\mathbf{x}^{(0)} + \alpha \mathbf{z}^{(1)})$ with $\alpha \in \{0, \frac{1}{2}, 1\}$. Then

$$\varphi(0) = 5, \quad \varphi\left(\frac{1}{2}\right) = 0.9093, \quad \varphi(1) = 5.9894$$

and so using the formula in Remark 1.3.1.2, we have

$$\alpha^* = \frac{3(5) + 5.9894 - 4(0.9093)}{4(5.9894 - 2(0.9093) + 5)} = 0.473$$

Hence, set $\alpha_1 = \alpha^*$. Update iterate

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_1 \mathbf{z}^{(1)} = \begin{bmatrix} 0.946 \\ 1.892 \end{bmatrix}$$

Now compute the gradient at $\mathbf{x}^{(1)}$, we have

$$\nabla g(\mathbf{x}^{(1)}) = \begin{bmatrix} -0.5191 \\ -0.4217 \end{bmatrix}$$

Step 2: Update directions.

- Fletcher–Reeves (FR):

$$\beta_1^{\mathrm{FR}} = \frac{\|\nabla g(\mathbf{x}^{(1)})\|^2}{\|\nabla g(\mathbf{x}^{(0)}\|^2} = 0.0224$$

  and update search direction

$$\mathbf{z}_{\mathrm{FR}}^{(2)} = -\nabla g(\mathbf{x}^{(1)}) + \beta_1^{\mathrm{FR}} \mathbf{z}^{(1)} = \begin{bmatrix} 0.5191 \\ 0.4217 \end{bmatrix} + 0.0224 \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 0.5639 \\ 0.5113 \end{bmatrix}$$

- Polak–Ribière (PR):

$$\beta_1^{\text{PR}} = \frac{\nabla g(\mathbf{x}^{(1)})^T(\nabla g(\mathbf{x}^{(1)}) - \nabla g(\mathbf{x}^{(0)}))}{\|\nabla g(\mathbf{x}^{(0)}\|^2}$$

Here, in the numerator,

$$\nabla g(\mathbf{x}^{(1)})^T(\nabla g(\mathbf{x}^{(1)}) - \nabla g(\mathbf{x}^{(0)})) = \begin{bmatrix} -2 \\ -4 \end{bmatrix}^T \begin{bmatrix} -2 + 0.5191 \\ -4 + 0.4217 \end{bmatrix} = \begin{bmatrix} 2 & 4 \end{bmatrix} \begin{bmatrix} -1.4809 \\ -3.5783 \end{bmatrix} = -17.275$$

and in the denominator, we have

$$\|\nabla g(\mathbf{x}^{(0)})\|^2 = (-2)^2 + (-4)^2 = 20$$

Hence,

$$\beta_1^{\text{PR}} = \frac{-17.275}{20} = -0.86375$$

In this case, $\max\{0, \beta_1^{\text{PR}}\} = 0$, so

$$\mathbf{z}_{\text{PR}}^{(2)} = -\nabla g(\mathbf{x}^{(1)}) = (0.5191, 0.4217)^T.$$

Nonlinear CG Method Code: `nonlinear_cg.py`

Chapter 3

Numerical Methods for Initial Value Problems of ODEs

Ordinary differential equations (ODEs) arise naturally in modeling time-dependent processes across physics, engineering, biology, and applied mathematics. In many cases, analytical solutions are either unavailable or impractical, making numerical methods essential for approximating solutions. This chapter focuses on numerical schemes for initial value problems (IVPs) of the form

$$\frac{dy}{dt} = f(y,t), \quad a \le t \le b, \quad y(a) = \alpha$$

The central task is to construct discrete approximations $Y_i \approx y(t_i)$ at a sequence of nodes $t_0 = a < t_1 < \cdots < t_N = b$ with step size $h = \frac{b-a}{N}$. A good numerical method balances accuracy, stability, and efficiency, while also being flexible enough to handle nonlinearities and systems of equations.

We begin with Euler's method, the simplex one-step scheme based on forward differences, and its natural extensions to high-order Taylor's methods, which exploit higher derivatives to improve accuracy. While conceptually straightforward, Taylor methods quickly become cumbersome in practice, motivating the development of derivative-free approaches.

Following section introduces the widely used Runge–Kutta methods, which achieve high-order accuracy without requiring explicit derivatives beyond the first. These methods form the foundation of most practical ODE solvers used in scientific computing.

In Section 3.3, we study multi-step methods, such as Adams–Bashforth and Adams–Moulton schemes, which reuse information from multiple past steps. These methods are computationally efficient for long-time integration and play a central role in modern solver packages.

A key concern in numerical ODEs is stability, discussed in Section 3.4. Stability analysis examines how numerical errors propagate, especially for stiff equations where naive methods may diverge despite small step sizes. Concepts such as absolute stability regions are introduced to evaluate and compare different methods.

Building on this, Section 3.5 presents adaptive step-size techniques, which dynamically adjust h to balance accuracy and efficiency. Adaptive methods are crucial in real-world problems where the solution may vary slowly over some intervals and rapidly over others.

Finally, in Section 3.6, we extend our discussion to systems of ODEs and higher-order equations, showing how they can be reformulated into first-order systems and tackled with the same numerical schemes. This highlights the generality and robustness of the methods developed in this chapter.

Through these sections, we establish both the theoretical foundations and the practical implementation of numerical methods for IVPs, preparing us to address more complex problems such as boundary value problems (Chapter 5) and applications in scientific computing.

3.1 The Euler's Method and High-Order Taylor Mehtods

Initial value problems of ordinary differential equations (ODEs) are central to modeling dynamical processes in science and engineering. Given an IVP of the form

$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha, \quad a \le t \le b,$$

the goal is to approximate the solution $y(t)$ at a discrete set of nodes $t_0 = a, t_1, \dots, t_N = b$. Since exact solutions are rarely available, numerical schemes are constructed by discretizing the differential equation into difference equations.

The Euler's method is the simplest and most fundamental one-step method. Derived from the forward difference approximation of the derivative, it generates the iterative scheme

$$Y_{i+1} = Y_i + hf(t_i, Y_i), \quad Y_0 = \alpha,$$

where h is the step size. Euler's method provides an intuitive geometric picture: at each step, the slope of the solution is approximated by the slope of the tangent line given by $f(t_i, Y_i)$. Despite its simplicity, Euler's method is only first-order accurate and often requires very small step sizes to achieve reliable results.

To improve accuracy, one can incorporate higher-order terms of the Taylor series expansion of the exact solution. The second-order Taylor method, for example, uses both $f(t, y)$ and its derivatives with respect to t and y to achieve a more accurate update rule. In general, the nth-order Taylor method produces an approximation with local truncation error of order $O(h^{n+1})$, significantly improving convergence compared to Euler's method.

While Taylor methods provide a systematic way to construct higher-order schemes, they often require computing complicated derivatives of $f(t, y)$, which can be impractical for nonlinear or large systems. Nevertheless, they play a crucial role in the theoretical development of numerical ODE solvers and form the foundation upon which more sophisticated derivative-free methods (such as Runge–Kutta methods) are built.

This section introduces Euler's method, analyzes its error behavior, and extends the idea to higher-order Taylor methods. We will also illustrate these schemes through worked examples, providing insight into both their accuracy and limitations.

3.1.1 Euler's Method

Consider the initial value problem (IVP) of a first-order ODE:

$$\frac{dy}{dt} = f(t, y), \quad a \le t \le b, \qquad y(a) = \alpha.$$

The goal is to approximate the solution $y(t)$ at discrete nodes

$$t_i = a + ih, \quad i = 0, 1, \ldots, N, \quad h = \frac{b-a}{N},$$

where h is the step size. We denote the numerical approximation at t_i by $Y_i \approx y(t_i)$.

From the differential equation, the derivative at t_i is

$$\frac{dy}{dt}(t_i) = f(t_i, y(t_i)).$$

Approximating the derivative by the forward difference,

$$\frac{dy}{dt}(t_i) \approx \frac{y(t_{i+1}) - y(t_i)}{h},$$

and replacing $y(t_i)$ by its numerical approximation Y_i, we obtain the scheme

$$Y_{i+1} = Y_i + hf(t_i, Y_i), \quad Y_0 = \alpha.$$

This recursive formula is known as the Euler's method. It provides a first-order explicit approximation to the true solution curve by following the slope given by the differential equation at each step.

Let $y(t)$ be the exact solution. From Taylor's expansion about t_i:

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \tfrac{1}{2}y''(\xi_i)h^2, \qquad t_i < \xi_i < t_{i+1}.$$

Since $y'(t_i) = f(t_i, y(t_i))$, we see that Euler's method approximates this expansion but omits the higher-order term. Thus, the local truncation error is

$$\tau_{i+1}(h) = \frac{y(t_{i+1}) - y(t_i)}{h} - f(t_i, y(t_i)) = \tfrac{1}{2}y''(\xi_i)h,$$

which satisfies

$$|\tau_{i+1}(h)| \le \tfrac{1}{2}Mh,$$

where $M = \max_{t\in[t_i,t_{i+1}]} |y''(t)|$. Hence, Euler's method is first-order accurate.

Theorem 3.1.1.1 (Global Error Estimate). *Suppose $f(t, y)$ is Lipschitz continuous in y with Lipschitz constant L. Then the global error satisfies*

$$|y(t_i) - Y_i| \le \frac{Mh}{2L}\left(e^{L(t_i-a)} - 1\right), \quad 1 \le i \le N,$$

showing that the method converges linearly as $h \to 0$.

Algorithm (Euler's Method)

1. Input: function $f(t, y)$, interval $[a, b]$, step size h, initial value α.
2. Set $t_0 = a$, $Y_0 = \alpha$, $N = \frac{b-a}{h}$.
3. For $i = 0, 1, \ldots, N - 1$:
$$Y_{i+1} = Y_i + hf(t_i, Y_i), \quad t_{i+1} = t_i + h$$
4. Output: (t_i, Y_i) for $i = 0, \ldots, N$.

Example 3.1.1.2. Consider the IVP

$$\frac{dy}{dt} = y - t^2 + 1, \quad y(0) = 0.5, \quad 0 \le t \le 1,$$

with step size $h = 0.2$.

- Step 0: $Y_0 = 0.5$
- Step 1: $Y_1 = Y_0 + hf(t_0, Y_0) = 0.5 + 0.2(0.5 - 0^2 + 1) = 0.8$.
- Further steps (Y_2, Y_3, Y_4, Y_5) are computed iteratively using the same scheme.

3.1.2 High-Order Taylor Methods

While Euler's method is simple and intuitive, its first-order accuracy often makes it inefficient for problems requiring high precision. A natural extension is to exploit higher-order terms from the Taylor series expansion of the exact solution.

For the IVP

$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha$$

the exact solution satisfies, for $h > 0$,

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2!}y''(t_i) + \cdots + \frac{h^n}{n!}y^{(n)}(t_i) + R_{n+1}$$

where $R_{n+1} = \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(\xi_i)$ for some $\xi_i \in (t_i, t_{i+1})$. Replacing derivatives of y using the ODE and its successive differentiations, we construct an nth-order scheme.

Second-Order Taylor Method

Expanding up to $O(h^2)$,

$$y(t_{i+1}) = y(t_i) + hf(t_i, y_i) + \frac{h^2}{2!}\left(f_t(t_i, y_i) + f_y(t_i, y_i)f(t_i, y_i)\right) + O(h^3)$$

Thus, the numerical scheme is

$$Y_{i+1} = Y_i + hT^{(2)}(t_i, y_i)$$

where

$$T^{(2)}(t, y) = f(t, y) + \frac{h}{2}\left(f_t(t, y) + f_y(t, y)f(t, y)\right)$$

The local truncation error satisfies

$$|\tau_{i+1}(h)| \leq \tfrac{M}{6}h^2,$$

so this method is second-order accurate.

General nth-Order Taylor Method

The nth-order Taylor method is defined as

$$Y_{i+1} = Y_i + hT^{(n)}(t_i, Y_i),$$

where

$$T^{(n)}(t,y) = f(t,y) + \frac{h}{2!}f^{[1]}(t,y) + \frac{h^2}{3!}f^{[2]}(t,y) + \cdots + \frac{h^{n-1}}{n!}f^{[n-1]}(t,y).$$

Here $f^{[k]}$ denotes the kth total derivative of f along the solution. The local truncation error is

$$|\tau_{i+1}(h)| \le \frac{M}{(n+1)!}h^n,$$

with $M = \max |y^{(n+1)}(t)|$ over $[t_i, t_{i+1}]$.

Thus the method achieves order n accuracy, reducing the global error to $O(h^n)$.

Example 3.1.2.1. Consider the IVP

$$\frac{dy}{dt} = y - t^2 + 1, \quad y(0) = 0.5, \quad 0 \le t \le 1,$$

with step size $h = 0.2$.

- Compute $f(t,y) = y - t^2 + 1$.
- Its partial derivatives are: $f_t(t,y) = -2t$ and $f_y(t,y) = 1$.
- Then
$$T^{(2)}(t,y) = f(t,y) + \frac{h}{2}\left(-2t + (1)(y - t^2 + 1)\right)$$

Applying this scheme step by step gives improved accuracy compared to Euler's method. For example, with $Y_0 = 0.5$:

$$Y_1 \approx 0.8293$$

vs. $Y_1 = 0.8$ from Euler's method.

Remark 3.1.2.2.

- **Advantages**
  - Higher accuracy than Euler's method.
  - Systematic extension to arbitrary order.
- **Limitations**
  - Requires computing derivatives of $f(t,y)$, which may be tedious or impractical for nonlinear or large systems.
  - Not commonly used in practice for high order; instead, Runge–Kutta methods achieve high accuracy without higher derivatives.

3.2 The Runge–Kutta Methods

The Runge–Kutta (RK) methods form a broad and powerful family of one-step methods for solving initial value problems (IVPs) of the form

$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha, \quad a \le t \le b$$

They were designed to achieve higher accuracy without the need to compute derivatives of f, unlike high-order Taylor methods.

From the second-order Taylor method

$$Y_{i+1} = Y_i + hf(t_i, Y_i) + \frac{h^2}{2}(f_t + f_y f)(t_i, Y_i)$$

the method requires the computation of f_t and f_y, which can be tedious.

The goal of Runge–Kutta schemes is to approximate these derivative terms using only function evaluations of f.

We seek a formula of the form

$$Y_{i+1} = Y_i + a_1 f(t_i, Y_i) + a_2 f(t_i + \alpha_2 h, y_i + \beta_2 h f(t_i, Y_i))$$

where the constants are chosen so that the local truncation error matches that of the second order Taylor expansion.

By matching coefficients via Taylor series expansion, one obtains

$$a_1 = 0, \quad a_2 = 1, \quad \alpha_2 = \frac{1}{2}, \quad \beta_2 = \frac{1}{2}$$

Hence,

$$Y_{i+1} = Y_i + hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2} f(t_i, Y_i)\right)$$

This is the Midpoint Method, a second order Runge–Kutta scheme with local truncation error $O(h^2)$.

Some other second order Runge–Kutta variants include

1. Modified Euler's Method

$$Y_{i+1} = Y_i + \frac{h}{2}[f(t_i, Y_i) + f(t_i + h, y_i + hf(t_i, Y_i))]$$

2. Heun's Method

$$Y_{i+1} = Y_i + \frac{h}{4}[f(t_i, Y_i) + 3f(t_i + \frac{2}{3}h, y_i + \frac{2}{3}hf(t_i, Y_i))]$$

All are second order methods with local truncation error $O(h^2)$.

Example 3.2.1. Solve

$$\frac{dy}{dt} = \frac{1}{5}\sin(y+t), \quad y(1) = 0.4, \quad 1 \le t \le 1.5$$

using the Modified Euler method with $h = 0.25$.

We have

$$Y_{i+1} = Y_i + \frac{h}{2}[f(t_i, Y_i) + f(t_i + h, y_i + hf(t_i, Y_i))]$$

with $t_0 = 1$ and $y_0 = 0.4$, and so,

$$Y_1 = 0.4 + 0.125\left[\frac{1}{5}\sin(1.4) + \frac{1}{5}\sin(1.25 + 0.4 + 0.25\frac{1}{5}\sin(1.4))\right] = 0.4494$$

$$Y_2 = 0.4969$$

To achieve higher accuracy without computing derivatives, the fourth-order Runge–Kutta method uses four intermediate slopes

$$\begin{aligned}
k_1 &= hf(t_i, Y_i)\\
k_2 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)\\
k_3 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)\\
k_4 &= hf\left(t_i + h, y_i + hk_3\right)
\end{aligned}$$

Then update

$$Y_{i+1} = Y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

The local truncation error is $O(h^4)$ and the method has global accuracy of $O(h^4)$.

Theorem 3.2.2 (Local Truncation Error for RK4). *For the classical RK4 step*

$$\begin{aligned}
k_1 &= hf(t_i, Y_i),\\
k_2 &= hf\left(t_i + \frac{h}{2}, Y_i + h\frac{k_1}{2}\right),\\
k_3 &= hf\left(t_i + \frac{h}{2}, Y_i + h\frac{k_2}{2}\right),\\
k_4 &= hf(t_i + h, Y_i + k_3)\\
Y_{i+1} &= Y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

the local truncation error (LTE) is

$$\tau_{i+1}(h) = \frac{1}{90}h^4 y^{(5)}(\xi_i), \quad \xi_i \in (t_i, t_{i+1}).$$

Proof. Consider the initial value problem $\frac{dy}{dt} = f(t,y)$. Define the total derivative along the solution by

$$D = \frac{\partial}{\partial t} + f(t,y)\frac{\partial}{\partial y}$$

Then, along the exact solution,

$$y' = f =: F, \quad y'' = DF, \quad y^{(3)} = D^2F, \quad \ldots$$

Our LTE (per-step residual divided by h) is

$$\tau(h) = \frac{1}{h}(y(t+h) - y(t)) - \frac{1}{6h}(k_1 + 2k_2 + 2k_3 + k_4)$$

evaluated with the exact $y(t)$ in the stages.

Using $y^{(m)} = D^{m-1}F$,

$$y(t+h) = y + hF + \frac{h^2}{2}DF + \frac{h^3}{6}D^2F + \frac{h^4}{24}D^3F + \frac{h^5}{120}D^4F + O(h^6).$$

So

$$\frac{1}{h}(y(t+h)-y(t)) = F + \frac{h}{2}DF + \frac{h^2}{6}D^2F + \frac{h^3}{24}D^3F + \frac{h^4}{120}D^4F + O(h^5) \quad (3.2.1)$$

We need to expand each stage value f at the perturbed points in powers of h along the flow. A compact way is to use that evaluating f at a point advanced by time αh along the exact flow corresponds (formally) to acting by $e^{\alpha hD}$ on F.

$$f(t+\alpha h, y(t+\alpha h)) = e^{\alpha hD}F = F + \alpha hDF + \frac{\alpha^2h^2}{2}D^2F + \frac{\alpha^3h^3}{6}D^3F + \frac{\alpha^4h^4}{24}D^4F + O(h^5)$$

In RK4, the internal states use forward Euler type predictions which are themselves expansions in h If you carry out the Taylor algebra carefully, you obtain the following stage expansions up to h^4

$$\begin{aligned}
k_1 &= hF, \\
k_2 &= h\left(F + \frac{h}{2}DF + \frac{h^2}{8}D^2F + \frac{h^3}{48}D^3F\right) + O(h^5), \\
k_3 &= h\left(F + \frac{h}{2}DF + \frac{h^2}{8}D^2F + \frac{h^3}{48}D^3F\right) + O(h^5), \\
k_4 &= h\left(F + hDF + \frac{h^2}{2}D^2F + \frac{h^3}{6}D^3F\right) + O(h^5).
\end{aligned}$$

Now average with RK4 weights

$$\frac{1}{6}(k_1+2k_2+2k_3+k_4) = h\left(F + \frac{h}{2}DF + \frac{h^2}{6}D^2F + \frac{h^3}{24}D^3F\right) + Ch^5D^4F + O(h^6)$$

where C is a constant we determine next.

A quick coefficient check shows that the terms through D^3F exactly match the Taylor series (3.2.1). That's the statement that RK4 is order 4. The first mismatch appears at order h^5, and a careful bookkeeping yields

$$C = \frac{1}{90}$$

Therefore,

$$\frac{1}{6}(k_1+2k_2+2k_3+k_4) = F + \frac{h}{2}DF + \frac{h^2}{6}D^2F + \frac{h^3}{24}D^3F + \frac{h^4}{90}D^4F + O(h^5) \tag{3.2.2}$$

Subtract (3.2.2) from (3.2.1), we get

$$\tau(h) = \left(\frac{1}{120} - \frac{1}{90}\right)h^4D^4F + O(h^5) = \frac{1}{90}h^4D^4F + O(h^5)$$

Since $D^4F = y^{(5)}$, we obtain

$$\tau_{i+1}(h) = \frac{1}{90}h^4y^{(t)}(\xi_i), \quad \xi_i \in (t_i, t_{i+1})$$

as desired. □

Remark 3.2.3. Euler's and Taylor's methods are explicit one-step methods, but Taylor requires partial derivatives.

Remark 3.2.4. Runge–Kutta methods are also explicit one-step methods, yet avoid derivative computations by combining multiple evaluations of f.

Remark 3.2.5. The classical RK4 is widely used in practice for its balance between accuracy and computational effort.

3.3 The Multi-Step Methods

So far, the methods we have studied—Euler's, Taylor's, and Runge–Kutta—are all one-step methods, meaning that each new approximation Y_{i+1} depends only on the immediately previous value Y_i.

However, in many situations, we can improve efficiency and accuracy by constructing multi-step methods, where Y_{i+1} depends on several previous points, i.e. $Y_i, Y_{i-1}, ..., Y_{i+1-m}$.

These methods are particularly useful when previous computed values are retained, thus reducing the number of function evaluations per step compared to high-order one-step methods.

Consider the initial value problem

$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha, \quad a \leq t \leq b$$

Integrating both sides over $[t_i, t_{i+1}]$, we have

$$y_{i+1} - y_i = \int_{t_i}^{t_{i+1}} f(t, y)\ dt$$

To approximate this integral numerically, we interpolate $f(t, y)$ using values at previous steps.

3.3.1 The Adams–Bashforth Two-Step Method (Explicit)

We approximate $f(t, y)$ on $[t_{i-1}, t_i]$ by a first degree polynomial $P_1(t)$ that passes through $(t_{i-1}, f(t_{i-1}, Y_{i-1}))$ and $(t_i, f(t_i, Y_i))$. Thus,

$$P_1(t) = (t_i - t)\frac{1}{h}f(t_{i-1}, Y_{i-1}) + (t - t_{i-1})\frac{1}{h}f(t_i, Y_i)$$

Substituting this into the integral form gives

$$Y_{i+1} - Y_i = \int_{t_i}^{t_{i+1}} P_1(t)\ dt = \frac{h}{2}[3f(t_i, Y_i) - f(t_{i-1}, Y_{i-1})].$$

Hence, the two-step Adams–Bashforth (AB2) method is

$$Y_{i+1} = Y_i = \frac{h}{2}[3f(t_i, Y_i) - f(t_{i-1}, Y_{i-1})]$$

Theorem 3.3.1.1 (Local Truncation Error for AB2). *The local truncation error for AB2 is given by*

$$\tau_{i+1}(h) = \frac{5}{12}h^2 y^{(3)}(\eta), \quad \eta \in (t_{i-1}, t_{i+1})$$

In other words, the AB2 method is second-order accurate with $O(h^2)$ local truncation error.

Proof. The local truncation error is what you get when you plug the exact solution $y(t)$ into the scheme and measure the residue per step

$$\tau_{i+1} = \frac{1}{h}(y(t_{i+1}) - y(t_i)) - \frac{1}{2}(3f(t_i, y(t_i)) - f(t_{i-1}, y(t_{i-1})))$$

Since $f(t, y(t)) = y'(t)$, this simplifies to

$$\tau_{i+1} = \frac{1}{h}(y_{i+1} - y_i) - \frac{1}{2}(3y'_i - y'_{i-1}),$$

where we write $y_i = y(t_i)$, $y'_i = y'(t_i)$, etc.

Expand about t_i assuming that y is C^3. For $y(t_{i+1})$,

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2}y''_i + \frac{h^3}{6}y^{(3)}(\xi_1), \quad \xi_1 \in (t_i, t_{i+1})$$

Hence,

$$\frac{1}{h}(y_{i+1} - y_i) = y'_i + \frac{h}{2}y''_i + \frac{h^2}{6}y^{(3)}(\xi_1)$$

For $y'(t_{i-1})$ (expand y' about t_i):

$$y'_{i-1} = y'_i - hy''_i + \frac{h^2}{2}y^{(3)}(\xi_2), \quad \xi_2 \in (t_{i-1}, t_i)$$

Substituting into the local truncation error expression,

$$\begin{aligned}
\tau_{i+1} &= \left(y'_i + \frac{h}{2}y''_i + \frac{h^2}{6}y^{(3)}(\xi_1)\right) - \frac{1}{2}\left(3y'_i - (y'_i + hy''_i + \frac{h}{2}y^{(3)}(\xi_2)\right) \\
&= \left(y'_i + \frac{h}{2}y''_i + \frac{h^2}{6}y^{(3)}(\xi_1)\right) - \left(y'_i + \frac{h}{2}y''_i - \frac{h^2}{4}y^{(3)}(\xi_2)\right) \\
&= \frac{h^2}{6}y^{(3)}(\xi_1) + \frac{h^2}{4}y^{(3)}(\xi_2) \\
&= h^2\left(\frac{1}{6}y^{(3)}(\xi_1) + \frac{1}{4}y^{(3)}(\xi_2)\right) \\
&= \frac{h^2}{12}\left(2y^{(3)}(\xi_1) + 3y^{(3)}(\xi_2)\right)
\end{aligned}$$

Since $y^{(3)}$ is continuous on $[t_{i-1}, t_{i+1}]$, by the Intermediate Value Theorem, there exists $\eta \in (t_{i-1}, t_{i+1})$ such that

$$2y^{(3)}(\xi_1) + 3y^{(3)}(\xi_2) = 5y^{(3)}(\eta)$$

Therefore,

$$\tau_{i+1} = \frac{5}{!2}h^2 y^{(3)}(\eta), \quad t_{i-1} < \eta < t_{i+1}$$

which shows the local truncation error is $O(h^2)$, and AB2 is second-order accurate. □

3.3.2 Euler's Method Revisited

We approximate $f(t, y(t))$ on $[t_i, t_{i+1}]$ by the constant value at the left endpoint

$$P_0(t) = f(t_i, Y_i)$$

Then

$$\int_{t_i}^{t_{i+1}} f(t, y(t))\, dt \approx \int_{t_i}^{t_{i+1}} P_0(t)\, dt = hf(t_i, Y_i)$$

Plug into the integral equation and repalce $y(t_i) \approx Y_i$, $y(t_{i+1}) \approx Y_{i+1}$,

$$Y_{i+1} - Y_i = hf(t_i, Y_i) \implies Y_{i+1} = Y_i + hf(t_i, Y_i)$$

This is explicit because the right hand side depends only on data at t_i. It is the quadrature left-rectangle rule applied to f, hence, first order. Indeed, the left-rectangle rule has quadrature error

$$\int_{t_i}^{t_{i+1}} f(t, y(t))\, dt = hf(t_i, y(t_i)) + \frac{h^2}{2}(f_t + f_y f)(\xi)$$

so the per-step defect is $O(h^2)$, and the local truncation error (defected divided by h) is $O(h)$.

3.3.3 Trapezoidal Method

We approximate $f(t, y(t))$ on $[t_i, t_{i+1}]$ by the line through its values at the endpoints $P_1(t)$ such that $P_1(t_i) = f(t_i, Y_i)$ and $P_1(t_{i+1}) = f(t_{i+1}, Y_{i+1})$. The integral of a linear function over an interval is the trapezoidal rule

$$\int_{t_i}^{t_{i+1}} f(t, y(t))\, dt \approx \int_{t_i}^{t_{i+1}} P_1(t)\, dt = \frac{h}{2}\left(f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})\right)$$

Insert this into the integral equation

$$Y_{i+1} - Y_i = \frac{h}{2}\left(f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})\right)$$

This is implicit because Y_{i+1} appears inside $f(t_{i+1}, Y_{i+1})$ but we must solve a (usually small) nonlinear equation each step.

The trapezoidal method is of second order because the trapezoid rule has quadrature error $O(h^3)$, so the per-step defect is $O(h^3)$ and the local truncation error is $O(h^2)$.

3.3.4 Adams–Moulton Two-Step Method (Implicit)

Using a quadratic interpolation $P_2(t)$ through three points

$$(t_{i-1}, f(t_{i-1}, Y_{i-1})), \quad (t_i, f(t_i, Y_i)), \quad (t_{i+1}, f(t_{i+1}, Y_{i+1}))$$

and integrating, we get

$$Y_{i+1} = Y_i + \frac{h}{12}\left(5f(t_{i+1}, Y_{i+1} + 8f(t_i, Y_i) - f(t_{i-1}, Y_{i-1})\right)$$

for all $i \geq 1$. This is known as the Adams–Moulton (AM2) method, which is implicit, but third-order accurate.

Theorem 3.3.4.1 (Local Truncation Error for AM2). *The local truncation error for AM2 is*

$$\tau_{i+1}(h) = -\frac{1}{24}h^3 y^{(4)}(\xi), \quad \xi \in (t_{i-1}, t_{i+1})$$

Proof. By definition, (plugging the exact solution y into the scheme and dividing by h) gives

$$\tau_{i+1}(h) = \frac{1}{h}(y_{i+1} - y_i) - \frac{1}{12}(5y'_{i+1} + 8y'_i - y'_{i-1})$$

where $y_k = y(t_k)$, $y'_k = y'(t_k)$, and $t_k = t_i + kh$ with uniform step h. We will expand everything about t_i using Taylor series.

For the function y,

$$\begin{aligned} y_{i+1} &= y_i + hy'_i + \frac{h^2}{2}y''_i + \frac{h^3}{6}y_i^{(3)} + \frac{h^4}{24}y_i^{(4)} + \frac{h^5}{120}y^{(5)}(\zeta_1) \\ y_{i-1} &= y_i - hy'_i + \frac{h^2}{2}y''_i 0\frac{h^3}{6}y_i^{(3)} + \frac{h^4}{24}y_i^{(4)} + \frac{h^5}{120}y^{(5)}(\zeta_2) \end{aligned}$$

and for the derivative y',

$$\begin{aligned} y'_{i+1} &= y'_i + hy''_i + \frac{h^2}{2}y_i^{(3)} + \frac{h^3}{6}y_i^{(4)} + \frac{h^4}{24}y^{(5)}(\eta_1) \\ y'_{i-1} &= y'_i - hy''_i + \frac{h^2}{2}y_i^{(3)} - \frac{h^3}{6}y_i^{(4)} + \frac{h^4}{24}y^{(5)}(\eta_2) \end{aligned}$$

We expand the two parts of $\tau_{i+1}(h)$ as follows. For the first part, we have

$$\frac{1}{h}(y_{i+1} - y_i) = y_i' + \frac{h}{2}y_i'' + \frac{h^2}{6}y_i^{(3)} + \frac{h^3}{24}y_i^{(4)} + \frac{h^4}{120}y^{(5)}(\zeta_1)$$

For the second piece, we form the combination inside the brackets

$$5y_{i+1}' + 8y_i' - y_{i-1}'$$

Using the expansions above, we collect like terms:

- y_i': $5 + 8 - 1 = 12$,
- y_i'': $5h - (-h) = 6h$,
- $y_i^{(3)}$: $5\frac{h^2}{2} - \frac{h^2}{2} = 2h^2$,
- $y_i^{(4)}$: $5\frac{h^3}{6} + \frac{h^3}{6} = h^3$,
- $y_i^{(5)}$: $5\frac{h^4}{24} - \frac{h^4}{24} = \frac{h^4}{6}$.

Thus,

$$\frac{1}{12}(5y_{i+1}' + 8y' - y_{i-1}') = y_i' + \frac{h}{2}y_i'' + \frac{h^2}{6}y_i^{(3)} + \frac{h^3}{12}y_i^{(4)} + \frac{h^4}{72}y^{(5)}(\tilde{\eta})$$

Now we subtract both parts

$$\begin{aligned}\tau_{i+1}(h) &= \left(y_i' + \frac{h}{2}y_i'' + \frac{h^2}{6}y_i^{(3)} + \frac{h^3}{24}y_i^{(4)} + \frac{h^4}{120}y^{(5)}(\zeta_1)\right)\\ &\quad - \left(y_i' + \frac{h}{2}y_i'' + \frac{h^2}{6}y_i^{(3)} + \frac{h^3}{12}y_i^{(4)} + \frac{h^4}{72}y^{(5)}(\tilde{\eta})\right)\\ &= \left(\frac{1}{24} - \frac{1}{2}\right)h^3y_i^{(4)} + \left(\frac{1}{120} - \frac{1}{72}\right)h^4y^{(5)}(\hat{\eta})\\ &= -\frac{1}{24}h^3y_i^{(4)} + O(h^4)\end{aligned}$$

where we have absorbed the various points into $\hat{\eta}$ and used boundedness and continuity of $y^{(5)}$.

Finally, by the Mean Value form of the remainder (and continuity of $y^{(4)}$), we replace $y_i^{(4)}$ by $y^{(4)}(\xi)$ for some $\xi \in (t_{i-1}, t_{i+1})$, obtaining the standard local truncation error formula

$$\tau_{i+1}(h) = -\frac{1}{24}h^3y^{(4)}(\xi), \quad \xi \in (t_{i-1}, t_{i+1}).$$

□

3.3.5 General m-Step Method

For the initial value problem $\frac{dy}{dt} = f(t, y)$, a linear multistep method with m steps has the form

$$\sum_{j=0}^{m} \alpha_j Y_{i+1-j} = h \sum_{j=0}^{m} \beta_j f(t_{i+1-j}, Y_{i+1-j})$$

where $m \geq 1$ is the number of steps, $\alpha_1, \alpha_2, ..., \alpha_m, \beta_1, \beta_2, ..., \beta_m$ are constant coefficients with $\alpha_0 \neq 0$ (usually we set $\alpha_0 = 1$ by scaling), h is the step size, and $Y_i \approx y(t_i)$ are the approximations at $t_i = a + ih$.

Example 3.3.6. Consider the initial value problem

$$\frac{dy}{dt} = -y + t + 1, \quad y(0) = 1$$

and take $h = 0.1$. We compute the first two steps Y_0, Y_1, and Y_2 using the four different methods mentioned above. Thus, we require $t_0 = 0$, $t_1 = 0.1$, and $t_2 = 0.2$.

Euler's Method: We have approximation solution given by

$$Y_{i+1} = Y_i + hf(t_i, Y_i) = Y_i + h(-Y_i + t_i + 1)$$

for $i = 1, 2$ and $Y_0 = 1$. Then

$$\begin{aligned}
&Y_0 = 1 \\
&Y_1 = Y_0 + h(-Y_0 + t_0 + 1) = 1 + 0.1(-1 + 0 + 1) = 1 \\
&Y_2 = Y_1 + h(-Y_1 + t_1 + 1) = 1 + 0.1(-1 + 0.1 + 1) = 1.01
\end{aligned}$$

Trapezoidal Method: We have the approximation solution given by

$$\begin{aligned}
Y_{i+1} &= Y_i + \frac{h}{2}\left(f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})\right) \\
&= Y_i + \frac{h}{2}(-Y_i + t_i + 1 - Y_{i+1} + t_{i+1} + 1) \\
&= Y_i + \frac{h}{2}(-(Y_{i+1} + Y_i) + (t_i + t_{i+1}) + 2)
\end{aligned}$$

for $i = 1, 2$ and $Y_0 = 1$. Then

$$\begin{aligned}
Y_1 &= Y_0 + \frac{h}{2}(-(Y_1 + Y_0) + (t_0 + t_1) + 2) \\
Y_1 &= 1 + \frac{0.1}{2}(-Y_1 + 0.1 + 1) \\
Y_1 &= 1 - 0.05Y_1 + 0.005 + 0.05 \\
1.05Y_1 &= 1.055 \\
Y_1 &= 1.005
\end{aligned}$$

and also,

$$\begin{aligned}
Y_2 &= Y_1 + \frac{h}{2}(-(Y_2 + Y_1) + (t_1 + t_2) + 2) \\
Y_2 &= 1.005 + \frac{0.1}{2}(-Y_2 - 1.005 + 0.3 + 2) \\
Y_2 &= 1.005 + 0.05(1.295 - Y_2) \\
Y_2 &= 1.005 + 0.06475 - 0.05Y_2 \\
1.05Y_2 &= 1.06975 \\
Y_2 &= 1.019.
\end{aligned}$$

Adams–Bashforth 2-Step Method: We have the approximation solution given by

$$\begin{aligned}
Y_{i+1} &= Y_i + \frac{h}{2}[3f(t_i, Y_i) - f(t_{i-1}, Y_{i-1})] \\
&= Y_i + \frac{h}{2}[3(-Y_i + t_i + 1) - (-Y_{i-1} + t_{i-1} + 1)] \\
&= Y_i + \frac{h}{2}[-3Y_i + 3t_i + 3 + Y_{i-1} - t_{i-1} - 1] \\
&= Y_i + \frac{h}{2}[(Y_{i-1} - 3Y_i) + (3t_i - t_{i-1}) + 2]
\end{aligned}$$

for $i = 2$, and $Y_0 = 1$. Using Euler, we can simply use $Y_1 = 1$ (which is simple, but a bit crude). Then

$$\begin{aligned}
Y_2 &= Y_1 + \frac{h}{2}[(Y_0 - 3Y_1) + (3t_1 - t_0) + 2] \\
&= 1 + \frac{0.1}{2}[(1 - 3 \cdot 1) + (3 \cdot 0.1 - 0) + 2] \\
&= 1.015
\end{aligned}$$

Adams–Moulton 2-Step Method: We have the approximation solution given by

$$Y_{i+1} = Y_i + \frac{h}{12}\left(5f(t_{i+1}, Y_{i+1}) + 8f(t_i, Y_i) - f(t_{i-1}, Y_{i-1})\right)$$

We also require a starter $Y_1 = 1$ found by Euler's method. Then

$$\begin{aligned}
Y_2 &= Y_1 + \frac{h}{12}(5f(t_2, Y_2) + 8f(t_1, Y_1) - f(t_0, Y_0)) \\
Y_2 &= 1 + \frac{0.1}{12}(5(-Y_2 + t_2 + 1) + 8(-Y_1 + t_1 + 1) - (-Y_0 + t_0 + 1)) \\
Y_2 &= 1 + \frac{0.1}{12}(5(-Y_2 + 0.2 + 1) + 8(-1 + 0.1 + 1) - (-1 + 0 + 1)) \\
Y_2 &= -\frac{1}{24}Y_2 + \frac{317}{300} \\
\frac{1}{24}Y_2 &= \frac{317}{300} \\
Y_2 &= 1.0144
\end{aligned}$$

3.4 Stability Analysis

Stability analysis helps determine whether small perturbations in the initial conditions or rounding errors will grow or decay during the numerical integration of ordinary differential equations. It is fundamental for understanding why and when a numerical method produces reliable results.

We consider the initial value problem (IVP)

$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha, \quad a \le t \le b$$

The goal is to ensure that the numerical scheme not only approximates the true solution accurately, but also behaves stably under small perturbations.

A general one-step-method is written as

$$Y_{i+1} = Y_i + h\Phi(t_i, Y_i, h), \quad i = 0, 1, 2, ..., N$$

with $Y_0 = \alpha$, where $\Phi(t, y, h)$ is derived from $f(t, y)$ and possibly its derivatives.

The local truncation error is

$$\tau_{i+1}(h) = \frac{y_{i+1} - y_i}{h} - \Phi(t_i, y_i, h) = O(h^\beta)$$

where β is the order of local truncation error.

Definition 3.4.1. The method is said to be *consistent* if

$$\lim_{h\to 0} \tau_{i+1}(h) = 0$$

Definition 3.4.2. The method is said to be *convergent* if the numerical solution Y_i approaches the exact solution $y(t_i)$ as the step size decreases.

Definition 3.4.3. A method is *stable* if small changes in the initial data or round-off errors cause only small changes in all subsequent approximations Y_i. In other words,

$$\lim_{h\to 0} |Y_{i+1} - y_{i+1}| = 0$$

Definition 3.4.4. The method is said to be *stable* if small perturbations in the initial value produce only small changes in the computed solution.

Theorem 3.4.5 (One-Step Method Stability). *If $\Phi(t, y, h)$ is Lipschitz continuous in y, then*

1. *The method is stable.*
2. *The method is convergent if and only it is consistent, i.e.*

$$\Phi(t, y, 0) = f(t, y) \quad \textit{and} \quad \lim_{h\to 0} \tau_{i+1}(h) = 0.$$

3. *If $|\tau_i(h)| \leq \tau(h) = O(h^\beta)$, then the global error satisfies*

$$|Y_i - y_i| \leq \frac{\tau(h)}{L}(e^{L(t_i - a)} - 1)$$

where L is the Lipschitz constant.

Hence, the order of convergence is the order of local truncation error.

Example 3.4.6. Consider the modified Euler's method

$$Y_{i+1} = Y_i + \frac{h}{2}\left[f(t_i, Y_i) + f(t_{i+1}, Y_i + hf(t_i, Y_i))\right]$$

Define

$$\Phi(t, y, h) = \frac{1}{2}[f(t, y) + f(t + h, y + hf(t, y))]$$

We claim that Φ is y-Lipschitz continuous. Indeed, observe that for any y_1, y_2, we have

$$|\Phi(t, y_1, h) - \Phi(t, y_2, h)| \leq \left(L + \frac{1}{2}hL^2\right)|y_1 - y_2| = L^*|y_1 - y_2|$$

so $L^* = L + \frac{1}{2}L^2$ (for $h \leq 1$) is a Lipschitz constant. Hence, the modified Euler method is stable.

From the local truncation error $\tau_{i+1}(h) = O(h^2)$, the method is convergent of order 2, with global error

$$|y_i - Y_i| = O(h^2).$$

For a general m-step method,

$$\begin{aligned} Y_{i+1} &= a_{m-1}Y_i + a_{m-2}Y_i + \cdots + a_0 Y_{i+1-m} \\ &\quad + h\left[b_m f(t_{i+1}, Y_{i+1}) + b_{m-1} f(t_i, Y_i) + \cdots + b_0 f(t_{i+1-m}, Y_{i+1-m}\right] \end{aligned}$$

The local truncation error is defined by inserting the exact solution $y(t)$,

$$\tau_{i+1}(h) = \frac{y_{i+1} - \sum_{j=0}^{m-1} a_j y_{i-j}}{h} - \sum_{j=0}^{m} b_j f(t_{i+1-j}, y_{i+1-j})$$

The characteristic equation of the homogeneous part

$$P(\lambda) = \lambda^m - a_{m-1}\lambda^{m-1} - a_{m-2}\lambda^{m-2} - \cdots - a_0 = 0$$

Let $\lambda_1, \lambda_2, ..., \lambda_m$ be its roots.

Proposition 3.4.7. *Consider the characteristic equation of the homogeneous part. If all $|\lambda_i| \leq 1$ and any root with $|\lambda_i| = 1$ is simple (not repeated), then the method satisfies the Root Condition.*

There are three different types of stabilities:

- Strongly stable: root condition satisfied; $\lambda = 1$ is the only root in the unit circle.
- Weakly stable: Root condition satisfied, but multiple roots on the unit circle.
- Unstable: At least one root has roots outside of the unit circle.

Example 3.4.8. Consider the 4-Step Adams–Bashforth Method given by

$$Y_{i+1} = Y_i + \frac{h}{24}[55f_i - 59f_{i-1} + 39f_{i-2} - 9f_{i-3}]$$

Since $t_{i+1-m} = t_{i-3}$, then $i + 1 - m = i - 3$, so we get $m = 4$. The characteristic equation is

$$P(\lambda) = \lambda^4 - \lambda^3 = \lambda^3(\lambda - 1) = 0$$

Hence, $\lambda_{1,2,3} = 0$ and $\lambda_4 = 1$. Hence, the method is strongly stable.

Example 3.4.9. Consider the method given by

$$Y_{i+1} = 4Y_i - 3Y_{i-1} - 2hf(t_i, Y_i)$$

Since $t_{i+1-m} = t_{i-1}$ then we get $m = 2$. The characteristic equation is

$$P(\lambda) = \lambda^2 - 4\lambda + 3 = (\lambda - 1)(\lambda - 3) = 0$$

Hence, $\lambda_1 = 1$ and $\lambda_2 = 3$. Since one root exceeds 1, the method is unstable.

The truncation error is

$$\tau_{i+1}(h) = \frac{y_{i+1} - 4y_i + 3y_{i-1}}{h} + 2f(t_i, y_i)$$

Since $y_{i+1} = y_i + hy'(t_i) + \frac{1}{2}h^2y''(\xi_1)$ where $\xi_1 \in (t_i, t_{i+1})$, and

$$y_{i-1} = y_i - hy'(t_i) + \frac{1}{2!}(-h)^2y''(\xi_2)$$

where $\xi_2 \in (t_{i-1}, t_i)$, then we have

$$\begin{aligned} y_{i+1} - 4y_i + 3y_{i-1} &= -2hy'(t_i) + \frac{1}{2}h^2(y''(\xi_1) + 3y''(\xi_2)) \\ &= -2hy'(t_i) + 2h^2y''(\xi) \end{aligned}$$

where $\xi \in (t_{i-1}, t_{i+1})$. So,

$$\tau_{i+1}(h) = -2y'(t_i) + 2hy''(\xi) + 2f(t_i, y_i) = 2hy''(\xi)$$

Thus,

$$|\tau_{i+1}(h)| \leq 2Mh$$

where $M = \max_{t\in[t_{i-1},t_{i+1}]} |y''(t)|$. Thus, the method is unstable, but its local truncation error is first order.

3.5 Adaptive Step Techniques

In earlier sections, we used numerical methods with fixed step sizes h (i.e. Euler, Runge–Kutta, Adams–Bashforth). However, in many real-world problems, the solution $y(t)$ may change rapidly in some regions and slowly in others.

Using a constant step size can be inefficient (too small a step wastes computation, while too large a step risks instability or loss of accuracy).

To address this, adaptive step size control adjusts h dynamically during integration to maintain a specified accuracy tolerance ε.

We consider the initial value problem (IVP)

$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha, \quad a \le t \le b.$$

Assume a numerical scheme of order n:

$$Y_{i+1} = Y_i + h\Phi(t_i, Y_i), \quad Y_0 = \alpha \tag{3.5.1}$$

with local truncation error

$$\tau_{i+1}(h) = O(h^n)$$

Given a tolerance ε, we want to choose a new step size h_{new} so that the local truncation error satisfies $|\tau_{i+1}(h)| \le \varepsilon$.

However, the constant M in $|\tau_{i+1}(h)| \le Mh^n$ is unknown for most ODEs. Therefore, we approximate the local truncation error using two methods of different orders.

Let

- Scheme (3.5.1): nth order method

$$Y_{i+1} = Y_i = h\Phi(t_i, Y_i),$$

- Scheme (3.5.2): $(n+1)$th order method

$$\bar{Y}_{i+1} = \bar{Y}_i + h\Psi(t_i, \bar{Y}_i) \tag{3.5.2}$$

Both start from the same point $Y_i = \bar{Y}_i = y_i$. Then

$$\tau_{i+1}(h) = \frac{1}{h}[y_{i+1} - Y_{i+1}], \quad \bar{\tau}_{i+1}(h) = \frac{1}{h}[y_{i+1} - \bar{Y}_{i+1}]$$

Subtracting gives

$$\tau_{i+1}(h) = \bar{\tau}_{i+1}(h) + \frac{1}{h}(\bar{Y}_{i+1} - Y_{i+1}).$$

Since $\bar{\tau}_{i+1}(h) = O(h^{n+1})$, we approximate

$$\tau_{i+1}(h) \approx \frac{1}{h}[\bar{Y}_{i+1} - Y_{i+1}]$$

Thus, the difference between the two approximations estimates the local error.

Since $|\tau_{i+1}(h)| \approx Kh^n$, we define a new step size $h_{\text{new}} = qh$. Then

$$|\tau_{i+1}(h_{\text{new}})| = K(qh)^n = q^n|\tau_{i+1}(h)|$$

To enforce $|\tau_{i+1}(h)| \leq \varepsilon$,

$$q^n \leq \frac{\varepsilon}{|\tau_{i+1}(h)|} = \frac{\varepsilon h}{|\bar{Y}_{i+1} - Y_{i+1}|}$$

Hence,

$$q = \left(\frac{\varepsilon h}{|\bar{Y}_{i+1} - Y_{i+1}|}\right)^{\frac{1}{n}}$$

To avoid drastic changes in step size, we restrict q by parameters l_0 and L_0 (i.e. $l_0 = \frac{1}{10}$ and $L_0 = 10$)

$$h_{\text{new}} = \begin{cases} l_0 h & \text{if } q \leq l_0 \\ qh & \text{if } l_0 < q < L_0 \\ L_0 h & \text{if } q \geq L_0 \end{cases}$$

Then we compute the next value using h_{new}

$$Y_{i+1} = Y_i + h_{\text{new}}\Phi(t_i, Y_i) \tag{3.5.3}$$

Example 3.5.1. We design an Adaptive Euler–Taylor method using

- Euler's method (order $n = 1$):

$$Y_{i+1} = Y_i + hf(t_i, Y_i)$$

- Taylor's method of order 2:

$$\bar{Y}_{i+1} = \bar{Y}_i + hT^2(t_i, \bar{Y}_i)$$

where

$$T^2(t, y) = f(t, y) + \frac{h}{2}(f_t + f_y f)(t, y)$$

Algorithm Steps:

1. Compute Y_{i+1} using Euler's method.
2. Compute $\bar{Y}_{i+1}$ using Taylor's method of order 2.
3. Estimate
$$q \approx \left(\frac{\varepsilon h}{|\bar{Y}_{i+1} - Y_{i+1}|} \right)^{\frac{1}{n}}$$
4. Define
$$h_{\text{new}} = \begin{cases} l_0 h & \text{if } q \leq l_0 \\ qh & \text{if } l_0 < q < L_0 \\ L_0 h & \text{if } q \geq L_0 \end{cases}$$
with $l_0 = 0.1$ and $L_0 = 10$
5. Recompute
$$Y_{i+1} = Y_i + h_{\text{new}} f(t_i, Y_i)$$
6. Continue to the next iteration.

Remark 3.5.2. The adaptive step approach balances accuracy and efficiency by dynamically adjusting h. The difference between two successive methods of adjacent order serves as an estimator of the local error.

For higher order methods, such as Runge–Kutta, embedded pairs such as RK4–RK5 are commonly used for adaptive control. For practical implementation, always limit the growth of h to prevent numerical instability.

3.6 Applications to Systems of ODEs and Higher-Order Equations

Up to now, we have developed numerical methods for first-order scalar IVPs of the form
$$\frac{dy}{dt} = f(t, y), \quad y(a) = \alpha.$$
In practice, most physical results such as mechanical vibrations, chemical kinetics, population dynamics, electrical circuits, etc, are governed by a systems of coupled ODEs or by higher order ODEs. Fortunately, the numerical techniques we studied extend naturally to these cases.

3.6.1 Systems of First Order ODEs

A general system of first-order ODEs can be written as

$$\begin{cases} y_1' = f_1(t, y_1, y_2, ..., y_n), \\ y_2' = f_2(t, y_1, y_2, ..., y_n), \\ \vdots \\ y_n' = f_n(t, y_1, y_2, ..., y_n), \end{cases} \qquad \mathbf{y}(a) = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

In vector notation,

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(a) = \alpha$$

where $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{f} : [a, b] \times \mathbb{R}^n \to \mathbb{R}^n$. All the scalar methods (Euler, Taylor, RK, multistep, etc.) can be applied componentwise using vector operations.

Example 3.6.1.1. We have Euler's method for Systems given by

$$\mathbf{Y}_{i+1} = \mathbf{Y}_i + h\mathbf{f}(t_i, \mathbf{Y}_i)$$

or component wise

$$Y_{j,i+1} = Y_{j,i} + hf_j(t_i, Y_{1,i}, Y_{2,i}, ..., Y_{n,i}), \quad j = 1, 2, ..., n.$$

Example 3.6.1.2. For each step

$$\begin{aligned} \mathbf{k}_1 &= h\mathbf{f}(t_i, \mathbf{Y}_i) \\ \mathbf{k}_2 &= h\mathbf{f}\left(t_i + \frac{h}{2}, \mathbf{Y}_i + \frac{\mathbf{k}_1}{2}\right) \\ \mathbf{k}_3 &= h\mathbf{f}\left(t_i + \frac{h}{2}, \mathbf{Y}_i + \frac{\mathbf{k}_2}{2}\right) \\ \mathbf{k}_4 &= h\mathbf{f}(t_i + h, \mathbf{Y}_i + \mathbf{k}_3) \end{aligned}$$

and thus, the RK4 method for systems of ODEs is given by

$$\mathbf{Y}_{i+1} = \mathbf{Y}_i + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4).$$

Each stage vector $\mathbf{k}_j \in \mathbb{R}^n$ and all vector additions and componentwise.

Example 3.6.1.3. Let $y_1(t)$ be the prey and $y_2(t)$ be the predator, and consider the system given by

$$\begin{cases} y_1' = 0.5y_1 - 0.02y_1y_2 \\ y_2' = -0.3y_2 + 0.01y_1y_2 \end{cases}$$

The interpretation is as follows:

- The prey grow exponentially at a rate of 0.5 when no predators are present; they are removed via encounters at a rate of $0.02y_1y_2$.
- The predators die at rate 0.3 without food; they increase via encounters at a rate of $0.01y_1y_2$.

This is the classic Lotka–Volterra family

$$\begin{cases} y_1' = ay_1 - by_1y_2 \\ y_2' = -cy_2 + dy_1y_2 \end{cases}$$

with $a = 0.5$, $b = 0.02$, $c = 0.3$, and $d = 0.01$.

The y_1-nullcline occurs when $y_1' = 0$, and thus, we find $y_1 = 0$ or $y_2 = 25$. On the other hand, the y_2-nullcline occurs when $y_2' = 0$, and thus, we find $y_2 = 0$ or $y_1 = 30$. The equilibria are their intersections: $(0,0)$ (which implies extinction), and $(30,25)$ (which implies coexistence). There are no other equilibria that exist, since, for example, $(30,0)$ is not an equilibrium since $y_1' \neq 0$ there.

Let $\mathbf{f} = (t, f_1, f_2)$. Then the Jacobian is

$$J(y_1,y_2) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 0.5 - 0.02y_2 & -0.02y_1 \\ 0.01y_2 & -0.3 + 0.01y_1 \end{bmatrix}$$

At $(0,0)$, the Jacobian becomes

$$J = \begin{bmatrix} 0.5 & 0 \\ 0 & -0.3 \end{bmatrix}$$

Hence, the eigenvalues are 0.5 and -0.3, so $(0,0)$ is an unstable saddle.

On the other hand, at $(30,25)$, the Jacobian becomes

$$J = \begin{bmatrix} 0 & -0.6 \\ 0.25 & 0 \end{bmatrix}$$

Hence, the eigenvalues are $\lambda = \pm i\sqrt{0.15}$, which is an imaginary pair, so the linearization is a center. For Lotka–Volterra, the nonlinear system actually has closed orbits surrounding $(30,25)$; solutions oscillate forever (no damping).

Lotka–Volterra systems possess a conserved quantity

$$H(y_1,y_2) = dy_1 - c\ln(y_1) + b_2 - a\ln(y_2)$$

so $\frac{d}{dt}H(y_1(t),y_2(t)) = 0$.

With $a = 0.5$, $b = 0.02$, $c = 0.3$, and $d = 0.01$, we have

$$H(y_1, y_2) = 0.01y_1 - 0.3\ln(y_1) + 0.02y_2 - 0.5\ln(y_2)$$

for $y_1, y_2 > 0$. Each initial condition $(y_1(0), y_2(0))$ selects a level set H constant.

We will apply the RK4 method to the predator-prey system as follows. For step size h, from $\mathbf{Y}_i = (Y_{1,i}, Y_{2,i})^T$,

$$\begin{aligned}
\mathbf{k}_1 &= h\mathbf{f}(t_i, \mathbf{Y}_i)\\
\mathbf{k}_2 &= h\mathbf{f}\left(t_i + \frac{h}{2}, \mathbf{Y}_i + \frac{\mathbf{k}_1}{2}\right)\\
\mathbf{k}_3 &= h\mathbf{f}\left(t_i + \frac{h}{2}, \mathbf{Y}_i + \frac{\mathbf{k}_2}{2}\right)\\
\mathbf{k}_4 &= h\mathbf{f}(t_i + h, \mathbf{Y}_i + \mathbf{k}_3)
\end{aligned}$$

Take $t_0 = 0$, $\mathbf{Y}_0 = (Y_{1,0}, Y_{2,0}) = (20, 30)$ and $h = 0.1$. Then

$$\mathbf{f}(y_1, y_2) = \begin{bmatrix} 0.5y_1 - 0.02y_1y_2 \\ -0.3y_2 + 0.01y_1y_1 \end{bmatrix}$$

Here,

$$\begin{aligned}
\mathbf{k}_1 &= h\mathbf{f}(0, 20, 30) = 0.1\begin{bmatrix}10 - 12\\ -9 + 6\end{bmatrix} = \begin{bmatrix}-0.2\\ -0.3\end{bmatrix}\\
\mathbf{k}_2 &= h\mathbf{f}\left(0.05, \mathbf{Y}_0 + \frac{\mathbf{k}_1}{2}\right) = h\mathbf{f}(0.05, 19.9, 29.85) \approx 0.1\begin{bmatrix}9.95 - 11.8665\\ -8.955 + 5.95015\end{bmatrix} = \begin{bmatrix}-0.19165\\ -0.300485\end{bmatrix}\\
\mathbf{k}_3 &= h\mathbf{f}\left(0.05, \mathbf{Y}_1 + \frac{\mathbf{k}_2}{2}\right) = h\mathbf{f}\left(0.05, 19.9042, 29.8498\right) \approx \begin{bmatrix}-0.19172\\ -0.30043\end{bmatrix}\\
\mathbf{k}_4 &\approx \begin{bmatrix}-0.18396\\ -3.0086\end{bmatrix}
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Y}_1 &= \begin{bmatrix}20\\ 30\end{bmatrix} + \frac{1}{6}\left(\begin{bmatrix}-0.2\\ -0.3\end{bmatrix} + 2\begin{bmatrix}-0.19165\\ -3.00485\end{bmatrix} + 2\begin{bmatrix}-0.19172\\ -0.30043\end{bmatrix} + \begin{bmatrix}-0.18396\\ -0.30086\end{bmatrix}\right)\\
&= \begin{bmatrix}19.8085\\ 29.6996\end{bmatrix}
\end{aligned}$$

3.6.2 Reduction of Higher-Order ODEs

Any mth order scalar ODE can be written as a first order system by introducing new variables up to order $m-1$. Consider

$$\frac{d^{(m)}y}{dt^{(m)}} = f(t, y, y', ..., y^{(m-1)}), \quad y(a) = \alpha_1, y'(a) = \alpha_2, ..., y^{(m-1)}(a) = \alpha_m$$

Define

$$y_1 = y, \quad y_2 = y', \quad ..., \quad y_m = y^{(m-1)}$$

Then

$$y_1' = y_2, \quad y_2' = y_3, \quad ..., \quad y_{m-1}' = y_m, \quad y_m' = f(t, y_1, ..., y_m)$$

which is a first order system of dimension m

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}).$$

Now we can apply any standard first-order ODE method.

Example 3.6.2.1. Consider the second order ODE given by

$$y'' = -2y' - y + t^2, \quad y(0) = 1, \quad y'(0) = 0$$

Then set $y_1 = y$ and $y_2 = y'$, so that

$$y_1' = y_2, \quad y_2' = -2y_2 - y_1 + t^2$$

Now apply, for example, Euler:

$$\mathbf{f}(t, \mathbf{y}^T) = \begin{bmatrix} y_2 \\ -2y_2 - y_1 + t^2 \end{bmatrix}$$

Start with $\mathbf{Y}_0 = (1, 0)^T$ with $h = 0.1$. Then note

$$\mathbf{f}(t_0, \mathbf{Y}_0^T) = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

and so

$$\mathbf{Y}_1 = \mathbf{Y}_0 + h\mathbf{f}(t_0, \mathbf{Y}_0^T) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.1 \end{bmatrix}$$

Example 3.6.2.2. Consider the harmonic oscillator given by

$$y'' + \omega^2 y = 0, \quad y(0) = 1, \quad y'(0) = 0.$$

Define $y_1 = y$ and $y_2 = y'$ so that $y_1' = y_2$ and $y_2' = -\omega^2 y_1$. Then

$$\mathbf{y}' = \begin{bmatrix} y_2 \\ -\omega^2 y_1 \end{bmatrix}$$

and can be solved via Euler, RK, or other adaptive schemes.

Adaptive techniques extend naturally to systems because both Y_{i+1} and $\bar{Y}_{i+1}$ are vectors. Compute the vector error estimate

$$\mathbf{e}_{i+1} = \bar{\mathbf{Y}}_{i+1} - \mathbf{Y}_{i+1}$$

and use its norm to control the step

$$q = \left(\frac{\varepsilon h}{\|\mathbf{e}_{i+1}\|_2} \right)^{\frac{1}{n}}$$

where $h_{\text{new}} = \text{clamp}(qh, [l_0 h, L_0 h])$. This ensures uniform relative accuracy across all components.

Chapter 4

Approximation Theory

In scientific computing and applied mathematics, exact analytical solutions to problems are often either unavailable or too complex to obtain. Approximation theory provides systematic techniques to represent complicated functions by simpler, well-understood ones (typically polynomials or trigonometric series) while controlling the resulting error. Such approximations form the foundation for numerical integration, interpolation, spectral methods, and the solution of differential equations.

The central goal is to find a function $p_n(x)$ from a predescribed finite-dimensional space (often the space of all polynomials of degree at most n) that best approximates a given function $f(x)$ over an interval $[a, b]$ in some sense (such as the least-squares norm of the uniform (Chebyshev) norm. Orthogonality, weight functions, and projection concepts play key roles in determining optimal coefficients and approximation errors.

This chapter introduces the fundamental ideas of approximation theory and develops several important classes of approximation techniques that are widely used in numerical analysis.

Collectively, these topics demonstrate how approximation theory unifies concepts from linear algebra, calculus, and functional analysis to construct efficient numerical representations of complex functions. Each section emphasizes both theoretical understanding and practical computation, illustrating how approximations can be tailored to achieve desired accuracy and efficiency in real-world applications.

4.1 Orthogonal Polynomials and Least Squares Approximations

In many applications, we are given a continuous function $f(x)$ defined on a closed interval $[a, b]$ and wish to approximate it by a polynomial

$$p_n(x) \in \mathcal{P}_n(\mathbb{R}) = \text{span}\{1, x, x^2, ..., x^n\}$$

The objective is to find $p_n(x)$ that best approximates $f(x)$ in the least-squares sense (i.e. that minimizes the weighted mean-square error between f and p_n.

Let $\omega(x) > 0$ almost everywhere on $[a, b]$ be a weight function. We define the error functional

$$E(p) = \int_a^b \omega(x)[f(x) - p(x)]^2 \, dx$$

We seek $p^*(x) \in \mathcal{P}_n(\mathbb{R})$ such that

$$E(p^*) = \min_{p \in \mathcal{P}_n(\mathbb{R})} E(p)$$

Writing

$$p(x) = \sum_{i=0}^{n} \alpha_i \varphi_i(x)$$

where $\{\varphi_i(x)\}_{i=1}^n$ is a chosen polynomial basis (commonly $\varphi_i(x) = x^i$), the minimization problem becomes

$$E(\boldsymbol{\alpha}) = \int_a^b \omega(x) \left[f(x) - \sum_{i=0}^{n} \alpha_i \varphi_i(x) \right]^2 \, dx$$

Setting the partial derivatives with respect to each coefficient to zero gives the normal equations

$$\frac{\partial E}{\partial a_i} = 0, \quad i = 0, 1, 2, ..., n$$

This yields a linear system

$$A\boldsymbol{\alpha} = \mathbf{b}$$

where

$$a_{i,j} = \int_a^b \omega(x)\varphi_i(x)\varphi_j(x) \, dx, \quad b_j = \int_a^b \omega(x) f(x) \varphi_j(x) \, dx$$

Solving for $\boldsymbol{\alpha}^* = A^{-1}\mathbf{b}$ gives

$$p^*(x) = \sum_{i=0}^{n} \alpha_i^* \varphi_i(x)$$

The computation of $\boldsymbol{\alpha}^*$ simplifies greatly if $\{\varphi_i\}_{i=0}^n$ is an orthogonal polynomial set with respect to $\omega(x)$, i.e.

$$\int_a^b \omega(x)\varphi_i(x)\varphi_j(x)\ dx = \begin{cases} 0 & \text{if } i \neq j \\ \beta_i > 0 & \text{if } i = j \end{cases}$$

In this case, the coefficient matrix A becomes diagonal

$$A = \text{diag}(\beta_0, \beta_1, ..., \beta_n)$$

and the coefficients are obtained independently

$$\alpha_i^* = \frac{b_i}{\beta_i} = \frac{\int_a^b \omega(x)f(x)\varphi_i(x)\ dx}{\int_a^b \omega(x)\varphi_i^2(x)\ dx}, \quad i = 0, 1, 2, ..., n$$

Hence, the least-squares approximation takes the compact form

$$p^*(x) = \sum_{i=0}^{n} \left(\frac{\int_a^b \omega(x)f(x)\varphi_i(x)\ dx}{\int_a^b \omega(x)\varphi_i^2(x)\ dx} \right) \varphi_i(x)$$

If an orthogonal set $\{\varphi_k(x)\}$ is not known a priori, it can be generated recursively by the three-term recurrence relation

$$\begin{aligned} &\varphi_0(x) = 1, \\ &\varphi_1(x) = (x - B_1)\varphi_0(x) \\ &\varphi_k(x) = (x - B_k)\varphi_{k-1}(x) - C_k\varphi_{k-2}(x), \quad 2 \leq k \leq n \end{aligned}$$

where

$$B_k = \frac{\int_a^b \omega(x)x\varphi_{k-1}^2(x)\ dx}{\int_a^b \omega(x)\varphi_{k-1}^2(x)\ dx}, \quad C_k = \frac{\int_a^b \omega(x)x\varphi_{k-1}(x)\varphi_{k-2}(x)\ dx}{\int_a^b \omega(x)\varphi_{k-2}^2(x)\ dx}$$

The resulting $\{\varphi_k(x)\}$ form an orthogonal polynomial system on $[a, b]$ with respect to $\omega(x)$.

Example 4.1.1 (Legendre Polynomials). We wish to construct an orthogonal polynomial system on the interval $[-1,1]$ with respect to the uniform weight function $\omega(x)=1$. That is, we seek polynomials $\{P_k(x)\}_{k=0}^n$ such that

$$\int_{-1}^{1} P_i(x)P_j(x)\ dx = \begin{cases} 0 & i \neq j \\ \int_{-1}^{1} P_i^2(x)\ dx > 0 & i = j \end{cases}$$

These are known as the Legendre polynomials. They play a fundamental role in continuous least squares approximation, as well as in numerical integration (Gauss–Legendre quadrature).

Start from the general three-term recurrence relation for constructing orthogonal polynomials

$$\begin{aligned}
&\varphi_0(x) = 1,\\
&\varphi_1(x) = (x - B_1)\varphi_0(x),\\
&\varphi_k(x) = (x - B_k)\varphi_{k-1}(x) - C_k\varphi_{k-2}(x), \quad k \geq 2
\end{aligned}$$

where the recurrence coefficients are defined by

$$B_k = \frac{\int_{-1}^{1}\omega(x)x\varphi_{k-1}^2(x)\ dx}{\int_{-1}^{1}\omega(x)\varphi_{k-1}^2(x)\ dx}, \quad C_k = \frac{\int_{-1}^{1}\omega(x)x\varphi_{k-1}(x)\varphi_{k-2}(x)\ dx}{\int_{-1}^{1}\omega(x)\varphi_{k-2}^2(x)\ dx}$$

For $\omega(x)=1$, the integral simplify to ordinary moments of powers of x on $[-1,1]$.

Now we compute the first few polynomial. For the zeroth polynomial, we have

$$P_0(x) = \varphi_0(x) = 1$$

For the first polynomial, we have

$$B_1 = \frac{\int_{-1}^{1} xP_0^2(x)\ dx}{\int_{-1}^{1} P_0^2(x)\ dx} = \frac{\int_{-1}^{1} x\ dx}{\int_{-1}^{1} 1\ dx} = \frac{0}{2} = 0$$

Hence,

$$P_1(x) = (x - B_1)P_0(x) = x$$

For the second polynomial, compute

$$B_2 = \frac{\int_{-1}^{1} xP_1^2(x)\ dx}{\int_{-1}^{1} P_1^2(x)\ dx} = \frac{\int_{-1}^{1} x^3\ dx}{\int_{-1}^{1} x^2\ dx} = \frac{0}{\frac{2}{3}} = 0.$$

and also,

$$C_2 = \frac{\int_{-1}^{1} xP_1(x)P_0(x)\ dx}{\int_{-1}^{1} P_0^2(x)\ dx} = \frac{\int_{-1}^{1} x^2}{\int_{-1}^{1} 1\ dx} = \frac{\frac{2}{3}}{2} = \frac{1}{3}$$

Then the recurrence gives

$$P_2(x) = (x - B_2)P_1(x) - C_2P_0(x) = x^2 - \frac{1}{3}$$

For the third polynomial, note that $B_3 = 0$ since $P_2(x)$ is even and $xP_2^2(x)$ is odd, so the integral in the numerator is 0. Also,

$$C_3 = \frac{\int_{-1}^{1} xP_2(x)P_1(x)\ dx}{\int_{-1}^{1} P_1^2(x)\ dx} = \frac{\int_{-1}^{1} x\left(x^2 - \frac{1}{3}\right) x\ dx}{\int_{-1}^{1} x^2\ dx} = \frac{\int_{-1}^{1}(x^4 - \frac{1}{3}x^2)\ dx}{\frac{2}{3}} = \frac{4}{15}$$

Therefore,

$$P_3(x) = (x - B_3)P_2(x) - C_3P_1(x) = x\left(x^2 - \frac{1}{3}\right) - \frac{4}{15}x = x^3 - \frac{3}{5}x$$

Thus, the first few Legendre polynomials are

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = x^2 - \frac{1}{3}, \quad P_3(x) = x^3 - \frac{3}{5}x$$

Now let us verify that P_0, P_1, and P_2 are orthogonal on $[-1, 1]$. Indeed,

$$\begin{aligned}
&\int_{-1}^{1} P_0(x)P_1(x)\ dx = \int_{-1}^{1} x\ dx = 0\\
&\int_{-1}^{1} P_0(x)P_2(x)\ dx = \int_{-1}^{1} \left(x^2 - \frac{1}{3}\right)\ dx = \frac{2}{3} - \frac{2}{3} = 0\\
&\int_{-1}^{1} P_1(x)P_2(x)\ dx = \int_{-1}^{1} x\left(x^2 - \frac{1}{3}\right)\ dx = \int_{-1}^{1} \left(x^3 - \frac{1}{3}x\right)\ dx = 0
\end{aligned}$$

Hence, they are mutually orthogonal.

The squared norms are given by

$$\|P_n\|_2^2 = \int_{-1}^{1} P_n^2(x)\ dx$$

For the first few, $\|P_0\|_2^2 = 2$, $\|P_1\|_2^2 = \frac{2}{3}$, and $\|P_2\|_2^2 = \frac{8}{45}$. In general, the Legendre polynomials satisfy the normalization formula

$$\int_{-1}^{1} P_n^2(x)\ dx = \frac{2}{2n+1}$$

Given $f(x)$ on $[-1,1]$, its continuous least-squares approximation (CLSA) of degree n using Legendre polynomials is

$$p_n(x) = \sum_{i=0}^{n} \alpha_i^* P_i(x)$$

where

$$\alpha_i^* = \frac{\int_{-1}^{1} f(x)P_i(x)\ dx}{\int_{-1}^{1} P_i^2(x)\ dx} = \frac{2i+1}{2}\int_{-1}^{1} f(x)P_i(x)\ dx$$

The second equality follows from the normalization formula.

Example 4.1.2. Let $f(x) = \sqrt{x}$ on $[1,2]$. We seek the quadratic least-squares approximation $p_2(x)$ with respect to the uniform weight $\omega(x) = 1$. Map $[1,2]$ to $[-1,1]$ using

$$\xi = \frac{2x-(a+b)}{b-a} = 2x-3 \implies x = \frac{1}{2}(\xi+3)$$

Define orthogonal polynomials on $[1,2]$ by transforminng the Legendre set

$$\varphi_k(x) = \left(\frac{b-a}{2}\right)^k P_k\left(\frac{2x-(a+b)}{b-a}\right), \quad k = 0,1,2$$

Now we compute

$$\alpha_i^* = \frac{\int_1^2 f(x)\varphi_i(x)\ dx}{\int_1^2 \varphi_i^2(x)\ dx}, \quad i = 0,1,2$$

Then $p_2(x) = \alpha_0^*\varphi_0(x) + \alpha_1^*\varphi_1(x) + \alpha_2^*\varphi_2(x)$ gives the quadratic least-square approximation to $f(x)$ on $[1,2]$.

4.2 Chebyshev Polynomials and Economization

This section focuses on Chebyshev polynomials, which minimize the maximum deviation between the approximating polynomial and the target function. We explore their extremal properties and the idea of economization, where a truncated Taylor series is replaced by a lower-degree Chebyshev approximation with comparable accuracy.

Definition 4.2.1. On the domain $[-1,1]$, the *Chebyshev polynomial of degree n* is defined as

$$T_n(x) = \cos(n \arccos(x)), \quad n \geq 0$$

Equivalently, $\{T_n(x)\}_{n=1}^{\infty}$ satisfies the recurrence relation

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

for $n \geq 1$.

Hence, the first few polynomials are

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2-1, \quad T_3(x) = 4x^3-3x, \quad T_4(x) = 8x^4-8x^2+1$$

Proposition 4.2.2. *The Chebyshev polynomials form an orthogonal set on* $(-1, 1)$ *with respect to the weight function*

$$\omega(x) = \frac{1}{\sqrt{1-x^2}}$$

That is,

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_n(x)T_m(x)\ dx = \begin{cases} 0 & \text{if } m \neq n \\ \frac{\pi}{2} & \text{if } m = n \neq 0 \\ \pi & \text{if } m = n = 0 \end{cases}$$

Proof. Set $x = \cos(\theta)$ with $\theta \in [0, \pi]$. Then $dx = -\sin(\theta)\ d\theta$, $\sqrt{1-x^2} = \sqrt{1-\cos^2(x)} = \sin(\theta)$, and

$$T_n(x)T_m(x) = \cos(n\theta)\cos(m\theta)$$

Hence,

$$\begin{aligned} \int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\ dx &= \int_{\pi}^{0} \frac{\cos(n\theta)\cos(m\theta)}{\sin(\theta)}(-\sin(\theta)\ d\theta) \\ &= \int_{0}^{\pi} \cos(n\theta)\cos(m\theta)\ d\theta \end{aligned}$$

So everything reduces to the standard cosine orthogonality on $[0, \pi]$. Using the identity

$$\cos(n\theta)\cos(m\theta) = \frac{1}{2}\left[\cos((n-m)\theta) + \cos((n+m)\theta)\right]$$

If $n \neq m$, then $n - m \neq 0$, and so

$$\int_{0}^{\pi} \cos((n-m)\theta)\ d\theta = \left[\frac{\sin((n-m)\theta)}{n-m}\right]_0^{\pi} = 0$$

and similarly, $\int_0^\pi \cos((n+m)\theta)\ d\theta = 0$. Thus, the integral is zero.

If $n = m \geq 1$, then

$$\int_0^\pi \cos^2(n\theta)\ d\theta = \int_0^\pi \frac{1}{2}(1+\cos(2n\theta))\ d\theta = \frac{1}{2}\left[\theta + \frac{\sin(2n\theta)}{2n}\right]_0^\pi = \frac{\pi}{2}$$

Finally, if $n = m = 0$, then

$$\int_0^\pi 1\ d\theta = \pi$$

Combining the three cases gives exactly

$$\int_{-1}^1 \frac{1}{\sqrt{1-x^2}} T_n(x)T_m(x)\ dx = \begin{cases} 0 & \text{if } m \neq n \\ \frac{\pi}{2} & \text{if } m = n \neq 0 \\ \pi & \text{if } m = n = 0 \end{cases}$$

□

This orthogonality makes $\{T_n(x)\}_{n=1}^\infty$ a natural basis for continuous least-squares approximation on $[-1,1]$.

Proposition 4.2.3. *For each $n \in \mathbb{N}_0$,*

1. *$T_n(x)$ has n simple roots $x_k = \cos\left(\frac{2k-1}{2n}\pi\right)$ for $k = 1, 2, ..., n$.*
2. *$T_n(x)$ has $n+1$ extreme points $x'_k = \cos\left(\frac{k\pi}{n}\right)$ for $k = 0, 1, 2, ..., n$ at which $T_n(x'_k) = (-1)^k$.*

Proof. Write $\theta = \arccos(x) \in [0, \pi]$ so that $x = \cos(\theta)$ and $T_n(x) = \cos(n\theta)$.

To see that (1) holds, note that $T_n(x) = 0$ if and only of $\cos(n\theta) = 0$. This holds exactly when

$$n\theta = \frac{\pi}{2} + m\pi \iff \theta = \frac{(2m+1)\pi}{2n}$$

for integers m. To keep $\theta \in (0, \pi)$, we take $m = 0, 1, ..., n-1$ producing n distinct angles

$$\theta_k = \frac{(2k-1)\pi}{2n}, \quad k = 1, 2, ..., n$$

and thus, n distinct zeros

$$x_k = \cos(\theta_k) = \cos\left(\frac{(2k-1)\pi}{2n}\right), \quad k = 1, 2, ..., n$$

For the simplicity of the roots, differentiate via θ

$$\frac{d}{dx}T_n(x) = \frac{d}{d\theta}\cos(n\theta)\frac{d\theta}{dx} = -n\sin(n\theta)\cdot\frac{1}{-\sin(\theta)} = \frac{n\sin(n\theta)}{\sin(\theta)}$$

At a zero, we have $n\theta = \frac{(2k-1)\pi}{2}$, so $\sin(n\theta) = \pm 1$, and $\sin(\theta) \neq 0$ for $\theta \in (0, \pi)$. Hence, $T_n'(x_k) \neq 0$, so each root is simple.

To see that (2) holds, critical points occur when $T_n'(x) = 0$. From the derivative above,

$$T_n'(x) = 0 \iff \sin(n\theta) = 0 \iff n\theta = k\pi$$

for $k = 0, 1, ..., n$ (these are exactly the solutions in $[0, \pi]$). Thus,

$$\theta_k' = \frac{k\pi}{n}, \quad x_k' = \cos(\theta_k') = \cos\left(\frac{k\pi}{n}\right), \quad k = 0, 1, ..., n$$

giving $n+1$ critical points in $[-1, 1]$. Evaluate T_n there so

$$T_n(x_k') = \cos(n\theta_k') = \cos(k\pi) = (-1)^k$$

Hence, these critical points are indeed extrema located at x_k' with the claimed alternating heights. The endpoints $k = 0$ and $k = n$ correspond to $x = 1$ and $x = -1$, respectively. □

These roots and extrema are crucial for interpolation and error minimization, forming the basis of Chebyshev nodes.

Proposition 4.2.4. *For a function $f : [-1, 1] \to \mathbb{R}$, the minimizing coefficients that minimize*

$$E(\alpha_0, \alpha_1, ..., \alpha_n) = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} \left[f(x) - \sum_{i=0}^{n} \alpha_i T_i(x)\right]^2 dx$$

are given by

$$\alpha_i^* = \frac{\int_{-1}^{1} \frac{f(x)T_i(x)}{\sqrt{1-x^2}}\, dx}{\int_{-1}^{1} \frac{T_i(x)^2}{\sqrt{1-x^2}}\, dx}, \quad i = 0, 1, 2, ..., n.$$

Furthermore, the continuous least-squares approximation is

$$p_n(x) = \sum_{i=0}^{n} \alpha_i^* T_i(x)$$

Proof. Define the weighted inner product and norm on polynomials (and more generally on L_2 with this weight) by

$$\langle g, h\rangle_w = \int_{-1}^{1} \frac{g(x)h(x)}{\sqrt{1-x^2}}\, dx, \quad \|g\|_w^2 = \langle g, g\rangle_w$$

It is elementary to verify that $\langle\cdot,\cdot\rangle_w$ is indeed an inner product.

Then

$$E(\alpha_0, \alpha_1, ..., \alpha_n) = \left\| f - \sum_{i=0}^{n} \alpha_i T_i \right\|_w^2$$

Since E is a strictly convex quadratic function of $(\alpha_0, \alpha_1, ..., \alpha_n)$ (its Hessian is the Gram matrix of $\{T_0, T_1, ..., T_n\}$ which is positive definite), the minimizer is characterized by the vanishing of the gradient

$$\frac{\partial E}{\partial \alpha_k} = -2\left\langle f - \sum_{i=0}^{n} \alpha_i T_i, T_k \right\rangle_w = 0, \quad k = 0, 1, 2, ..., n$$

Equivalently,

$$\sum_{i=0}^{n} \alpha_i \langle T_i, T_k\rangle_w = \langle f, T_k\rangle_w, \quad k = 0, 1, 2, ..., n$$

The Chebyshev polynomials $\{T_i\}$ are orthogonal with respect to the weight function $w(x) = \frac{1}{\sqrt{1-x^2}}$, so $\langle T_i, T_k\rangle_w = 0$ for $i \neq k$, while

$$\langle T_i, T_k\rangle_w = \int_{-1}^{1} \frac{T_i(x)^2}{\sqrt{1-x^2}}\, dx > 0$$

Hence, the system decouples diagonally and yields

$$\alpha_k^* = \frac{\langle f, T_k\rangle_w}{\langle T_k, T_k\rangle_w} = \frac{\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\, dx}{\int_{-1}^{1} \frac{T_k(x)^2}{\sqrt{1-x^2}}\, dx}, \quad k = 0, 1, 2, ..., n$$

Substituting these coefficients gives the minimizer

$$p_n(x) = \sum_{i=0}^{n} \alpha_i^*(x) T_i(x)$$

which is precisely the orthogonal projection of f onto span$\{T_0, T_1, ..., T_n\}$ in the weighted L_2 space, hence, the unique least-squares polynomial. □

Corollary 4.2.5. *Using*

$$\int_{-1}^{1} \frac{T_i(x)^2}{\sqrt{1-x^2}}\, dx = \begin{cases} \pi & \text{if } i = 0 \\ \frac{\pi}{2} & \text{if } i \geq 1 \end{cases}$$

then

$$\alpha_0^* = \frac{1}{\pi}\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\, dx, \quad \alpha_i^* = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_i(x)}{\sqrt{1-x^2}}\, dx \quad (i \geq 1)$$

These are the standard Chebyshev expansion coefficient.s

Proof. Let $x = \cos(\theta)$ with $\theta \in [0, \pi]$. Then

$$T_n(x) = \cos(n\theta), \quad dx = -\sin(\theta)\, d\theta, \quad \sqrt{1-x^2} = \sin(\theta)$$

Hence,

$$\int_{-1}^{1} \frac{T_n(x)^2}{\sqrt{1-x^2}}\, dx = \int_{\pi}^{0} \frac{\cos^2(n\theta)}{\sin(\theta)}(-\sin(\theta)\, d\theta) = \int_{0}^{\pi} \cos^2(n\theta)\, d\theta$$

Using $\cos^2(n\theta) = \frac{1}{2}(1+\cos(2n\theta))$,

- If $n = 0$, then
$$\int_0^{\pi} \cos^2(0 \cdot \theta)\, d\theta = \int_0^{\pi} 1\, d\theta = \pi$$
- If $n \geq 1$, then
$$\int_0^{\pi} \cos^2(n\theta)\, d\theta = \frac{1}{2}\int_0^{\pi} 1\, d\theta + \frac{1}{2}\int_0^{\pi} \cos(2n\theta)\, d\theta = \frac{1}{2}\pi = \frac{\pi}{2}$$

This gives the stated denominators. Plugging them into

$$\alpha_n^* = \frac{\int_{-1}^{1} \frac{f(x)T_n(x)}{\sqrt{1-x^2}}\, dx}{\int_{-1}^{1} \frac{T_n(x)^2}{\sqrt{1-x^2}}\, dx}$$

yields

$$\alpha_0^* = \frac{1}{\pi}\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\, dx, \quad \alpha_i^* = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_i(x)}{\sqrt{1-x^2}}\, dx \quad (i \geq 1)$$

as desired. □

Before discussing Chebyshev interpolation, it is useful to recall how polynomial interpolation works in general.

Definition 4.2.6. Given $n+1$ distinct points $(x_0, f(x_0))$, $(x_1, f(x_1))$, ..., $(x_n, f(x_n))$, the unique polynomial $P_n(x)$ of degree at most n that satisfies

$$P_n(x_i) = f(x_i), \quad i = 0, 1, 2, ..., n$$

is called the *Lagrange interpolating polynomial* for f at those nodes.

It can be written as

$$P_n(x) = \sum_{i=0}^{n} f(x_i)\ell_{n,i}(x)$$

where $\ell_{n,i}(x)$ are the Lagrange basis functions defined by

$$\ell_{n,i}(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Each $\ell_{n,i}(x)$ satisfies the Kronecker property $L_{n,i}(x_j) = \delta_{j,i}$, where

$$\delta_{j,i} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{if } j \neq i \end{cases}$$

Thus, $P_n(x)$ interpolates f exactly at all chosen nodes.

For $f \in C^{n+1}([-1,1])$, the interpolation error at any $x \in [-1,1]$ is given by

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j), \quad \text{for some } \xi \in (-1, 1)$$

Hence, the magnitude of the interpolation error depends both on the derivative $f^{(n+1)}$ and the distribution of the interpolation nodes x_j.

When nodes are equally spaced, $\prod_{j=0}^{n}(x - x_j)$ tends to oscillate violently near the endpoints (the Runge phenomenon). To control these oscillations, we must choose the nodes more carefully, leading to the use of Chebyshev nodes.

For interpolation, choose nodes $x_0, x_1, ..., x_n$ in $[-1, 1]$ and define

$$P_n(x) = \sum_{i=0}^{n} f(x_i)\ell_{n,i}(x), \quad \ell_{n,i}(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Then the interpolation error satisfies

$$|f(x) - P_n(x)| = \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \prod_{j=0}^{n} |x - x_j|$$

To minimize the maximum interpolation error, we must choose nodes that minimize

$$\max_{x\in[-1,1]} \prod_{j=0}^{n} |x - x_j|$$

Theorem 4.2.7. *Among all monic polynomials of degree n, the scaled Chebyshev polynomial*

$$\tilde{T}_n(x) = \frac{1}{2^{n-1}} T_n(x)$$

achieves the smallest possible maximum value on $[-1,1]$. That is, for all $P_n(x) \in \mathcal{P}_n^{\text{monic}}$,

$$\frac{1}{2^{n-1}} = \max_{x\in[-1,1]} |\tilde{T}_n(x)| \leq \max_{x\in[-1,1]} |P_n(x)|$$

where $\mathcal{P}_n^{\text{monic}}$ is the set of all monic polynomials of degree n.

Proof. Let $x'_k = \cos\left(\frac{k\pi}{n}\right)$ for $k = 0, 1, 2, ..., n$. Then

$$T_n(x'_k) = \cos\left(n \cdot \frac{k\pi}{n}\right) = \cos(k\pi) = (-1)^k$$

These are exactly the $n+1$ extrema of T_n on $[-1,1]$, and T_n is strictly monotone on each open interval (x'_{k+1}, x'_k).

For $n \geq 1$, the leading coefficient of T_n is 2^{n-1}. This can be proved by induction from the three-term recurrence

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

using that $T_0 = 1$ and $T_1 = x$. Hence, $\tilde{T}_n = \frac{1}{2^{n-1}} T_n$ is monic.

Since $|T_n(x)| \leq 1$ on $[-1,1]$ with equality at the extrema x'_k, we have

$$\|\tilde{T}_n\|_\infty = \frac{1}{2^{n-1}} \|T_n\|_\infty = \frac{1}{2^{n-1}}$$

Let $P \in \mathcal{P}_n^{\text{monic}}$ and set

$$Q(x) = \tilde{T}_n(x) - P(x)$$

Then $\deg(Q) \leq n-1$ (the leading x^n-terms cancel). Evaluate Q at the extrema $x'_k = \cos\left(\frac{k\pi}{n}\right)$. Since

$$\tilde{T}_n(x'_k) = \frac{1}{2^{n-1}}(-1)^k$$

we get

$$Q(x'_k) = \frac{(-1)^k}{2^{n-1}} - P(x'_k)$$

We consider the following cases.

Case 1 (If $\|P\|_\infty\| < \frac{1}{2^{n-1}}$): Then for each k,

$$(-1)^k Q(x'_k) = \frac{1}{2^{n-1}} - (-1)^k P(x'_k) > 0$$

i.e. $Q(x'_k)$ strictly alternates sign as k increases.

Case 2 (If $\|P\|_\infty\| \leq \frac{1}{2^{n-1}}$): Then for each k,

$$(-1)^k Q(x'_k) = \frac{1}{2^{n-1}} - (-1)^k P(x'_k) \geq 0$$

hence, $Q(x'_k)$ alternates non-positively/non-negatively (some values may be zero, but the parity flips the inequality at successive k).

In either case, between each consecutive pair $x'_{k+1} < x'_k$, the values $Q(x'_{k+1})$ and $Q(x'_k)$ are of opposite sign or at least one is zero. By the Intermediate Value Theorem and using that T_n (and hence, $\tilde{T}_n$) is strictly monotone on each (x'_{k+1}, x'_k), there exists at least one root of Q in every closed subinterval $[x'_{k+1}, x'_k]$. Therefore, Q has at least n distinct roots in $[-1, 1]$, which is a contradiction since $\deg(Q) \leq n-1$, unless Q is the zero polynomial. □

In practical computation, we often replace a high-degree Taylor polynomial by a lower-degree one that approximates $f(x)$ to nearly the same accuracy over $[-1, 1]$. This process is called Chebyshev economization.

Given a polynomial $P_n(x)$ such that

$$|f(x) - P_n(x)| \leq \frac{\varepsilon}{2}$$

we seek a lower-degree polynomial $P_{n-1}(x)$ satisfying

$$|f(x) - P_{n-1}(x)| \leq \varepsilon, \quad |P_n(x) - P_{n-1}(x)| \leq \frac{\varepsilon}{2}$$

By re-expanding $P_n(x)$ in Chebyshev form and dropping the smallest coefficients, we achieve a more economical polynomial with comparable accuracy.

Example 4.2.8. Let $f(x) = e^x$ on $[-1, 1]$ with $\varepsilon_0 = 0.05$. The Taylor series of about $x_0 = 0$ is

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{e^\xi x^5}{5!}, \quad |\xi| \le 1$$

Hence, the fourth-degree Taylor polynomial is

$$P_4(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}$$

The truncation error satisfies

$$\begin{aligned}|e^x - P_4(x)| &= \left|\left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{e^\xi x^5}{5!}\right) - \left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}\right)\right| \\ &= \left|\frac{e^\xi x^5}{5!}\right| \\ &\le \frac{e}{120} < 0.05\end{aligned}$$

Now we find $P_3(x)$ such that $|e^x - P_3(x)| \le 0.05$. It suffices to ensure that

$$|P_4(x) - P_3(x)| \le 0.025$$

Let $P_3(x) = P_4(x) - a_4\tilde{T}_4(x)$ where $a_4 = \frac{1}{4!}$. Here,

$$\begin{aligned}P_3(x) &= P_4(x) - \frac{1}{4! \cdot 2^3} T_4(x) \\ &= \left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}\right) - \frac{1}{192}\left(8x^4 - 8x^2 + 1\right) \\ &= \frac{191}{192} + x + \frac{13}{24}x^2 + \frac{1}{6}x^3\end{aligned}$$

Now observe

$$\begin{aligned}|P_4(x) - P_3(x)| &= |a_4\tilde{T}_4(x)| \\ &\le \frac{1}{24} \max_{x\in[-1,1]} |\tilde{T}_4(x)| \\ &= \frac{1}{24} \cdot \frac{1}{8} \\ &= \frac{1}{192}\end{aligned}$$

Finally,

$$\begin{aligned}|e^x - P_3(x)| &\le |e^x - P_4(x)| + |P_4(x) - P_3(x)|\\ &\le \frac{e}{120} + \frac{1}{192}\\ &= 0.0283\\ &< 0.05\end{aligned}$$

4.3 Fourier Approximation and Fast Fourier Transforms

Finally, we study periodic function approximations using Fourier series and describe the Fast Fourier Transform (FFT), an efficient algorithm for computing Fourier coefficients. These methods underpin modern signal processing, spectral methods for differential equations, and numerous applications in engineering and physics.

On the interval $[-\pi, \pi]$, define the trigonometric basis functions

$$\begin{aligned}\varphi_0(x) &= \frac{1}{2},\\ \varphi_k(x) &= \cos(kx), \quad k = 1, 2, ..., n,\\ \varphi_{n+k}(x) &= \sin(kx), \quad k = 1, 2, ..., n-1\end{aligned}$$

for $k = 1, 2, ..., n$. These $2n$ functions form the Fourier basis. Let the weight function be $\omega(x) = 1$. Then the set $\{\varphi_k(x)\}_{k=0}^{2n-1}$ is orthogonal on $[-\pi, \pi]$

$$\begin{aligned}\int_{-\pi}^{\pi} \varphi_k(x)\varphi_j(x)\ dx = 0, \quad k \neq j\\ \int_{-\pi}^{\pi} \varphi_j(x)^2\ dx = \pi, \quad k = j\end{aligned}$$

We seek the trigonometric polynomial of order n

$$S_n(x) = \sum_{k=0}^{2n-1} a_k \varphi_x(x)$$

that minimizes

$$E(a_0, a_1, ..., a_{2n-1}) = \int_{-\pi}^{\pi} [f(x) - S_n(x)]^2\ dx$$

Setting $\frac{\partial E}{\partial a_i} = 0$ gives

$$a_i^* = \frac{1}{\pi} \int_{-\pi}^{|pi} f(x)\varphi_i(x)\ dx, \quad i = 0, 1, 2, ..., 2n-1$$

Hence, the continuous Fourier approximation is

$$S_n(x) = \sum_{k=0}^{2n-1} a_k^* \varphi_k(x) \approx f(x)$$

In conventional notation,

$$S_n(x) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1} (a_k \cos(kx) + b_k \sin(kx))$$

where

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx)\ dx, \quad b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(kx)\ dx$$

Let $m \geq n$ and take $2m$ equally spaced nodes

$$x_j = -\pi + \frac{j\pi}{m}, \quad j = 0, 1, 2, ..., 2m-1$$

Then

$$E = \sum_{j=0}^{2m-1} [f(x_j) - S_n(x_j)]^2$$

To minimize E, use orthogonality of the discrete trigonometric system.

Theorem 4.3.1. *For $m > n$,*

$$\sum_{j=0}^{2m-1} \varphi_k(x_)\varphi_\ell(x_j) = \begin{cases} 0 & \text{if } k \neq \ell \\ m & \text{if } k = \ell \end{cases}$$

Sketch of Proof. Using the identities

$$\sum_{j=0}^{2m-1} e^{irx_j} = e^{-1r\pi} \frac{1 - e^{\frac{i2mr\pi}{m}}}{1 - e^{\frac{ir\pi}{m}}} = 0$$

when R is not a multiple of $2m$, we obtain

$$\sum_{j=0}^{2m-1} \cos(rx_j) = \sum_{j=0}^{2m-1} \sin(rx_j) = 0$$

and consequently, the stated orthogonality relations. □

Minimizing E with respect to a_k and b_k yields

$$a_k^* = \frac{1}{m}\sum_{j=0}^{2m-1} f(x_j)\cos(kx_j), \quad b_k^* = \frac{1}{m}\sum_{j=0}^{2m-1} f(x_j)\sin(kx_j)$$

for $k = 1, 2, ..., n-1$ and

$$a_0^* = \frac{1}{m}\sum_{j=0}^{2m-1} f(x_j), \quad a_n^* = \frac{1}{m}\sum_{j=0}^{2m-1} f(x_j)\cos(nx_j)$$

Therefore,

$$S_n(x) = \frac{a_0^*}{2} + a_n^*\cos(nx) + \sum_{k=1}^{n-1}(a_k^*\cos(kx) + b_k^*\sin(kx))$$

Lemma 4.3.2. *For integers r not multiples of m or $2m$,*

$$\sum_{j=0}^{2m-1}\cos(rx_j) = \sum_{j=0}^{2m-1}\sin(rx_j) = 0$$

and

$$\sum_{j=0}^{2m-1}(\cos(rx_j))^2 = \sum_{j=0}^{2m-1}(\sin(rx_j))^2 = m$$

Example 4.3.3. Let $f(x) = x$ on $[-\pi, \pi]$. Find $S_3(x)$ by continuous Fourier approximation.

We compute

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} x\cos(kx)\ dx, \quad b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} x\sin(x)\ dx$$

By symmetry, $a_k = 0$ for all k, and integration by parts gives

$$b_k = \frac{2(-1)^{k+1}}{k}$$

Hence,

$$S_3(x) = 2\sum_{k=1}^{3}\frac{(-1)^{k+1}}{k}\sin(kx) = 2\left(\sin(x) - \frac{1}{2}\sin(2x) + \frac{1}{3}\sin(3x)\right)$$

This approximates $f(x) = x$ on $[-\pi, \pi]$ in the least-squares sense.

Example 4.3.4. Let $f(x) = |x|$ on $[-\pi, \pi]$. Since f is even, its Fourier series has only cosine terms

$$|x| \sim \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(kx)$$

Since $|x|$ is even and $\cos(kx)$ is even,

$$a_k = \frac{2}{\pi} \int_0^{\pi} x \cos(kx)\ dx$$

Integration by parts (or a quick table integral) gives

$$\int_0^{\pi} x \cos(kx)\ dx = \frac{1-(-1)^k}{k^2}$$

Hence,

$$a_k = \frac{2}{\pi} \cdot \frac{1-(-1)^k}{k^2} = \begin{cases} \frac{4}{k^2\pi} & \text{if } k \text{ is odd} \\ 0 & \text{if } k \text{ is even} \end{cases}$$

Also,

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} |x|\ dx = \frac{2}{\pi} \int_0^{\pi} x\ dx = \pi$$

Therefore, the Fourier series of $|x|$ is

$$|x| = \frac{\pi}{2} + \frac{4}{\pi} \sum_{m=0}^{\infty} \frac{\cos((2m+1)x)}{(2m+1)^2}$$

which converges pointwise on $[-\pi, \pi]$. Furthermore, keeping the first n odd modes gives

$$S_n(x) = \frac{\pi}{2} + \frac{4}{\pi} \sum_{m=0}^{n} \frac{\cos((2m+1)x)}{(2m+1)^2}$$

The discrete Fourier coefficients

$$c_k = \sum_{j=0}^{2m-1} y_j e^{\frac{ik\pi j}{m}}, \quad k = 0, 1, 2, ..., 2m-1$$

require $O(4m^2)$ complex multiplications if evaluated directly. For large m, this becomes prohibitively expensive.

The Fast Fourier Transform (FFT) reduces the computational cost to $O(m \log_2(m))$ by exploiting symmetry and periodicity in $e^{\frac{ik\pi j}{m}}$.

Let $m = 2^p$ so that there are $2m = 2^{p+1}$ data points. We can separate even and odd indices in the summation

$$\begin{aligned} c_k &= \sum_{j=0}^{2m-1} y_j e^{\frac{ik\pi j}{m}} \\ &= \sum_{j=0}^{m-1} y_{2j} e^{\frac{ik\pi(2j)}{m}} + \sum_{j=0}^{m-1} y_{2j+1} e^{\frac{ik\pi(2j+1)}{m}} \\ &= \sum_{j=0}^{m-1} y_{2j} e^{\frac{ik\pi j}{\frac{m}{2}}} + e^{\frac{ik\pi}{m}} \sum_{j=0}^{m-1} y_{2j+1} e^{\frac{ik\pi j}{\frac{m}{2}}} \end{aligned}$$

Define

$$E_k = \sum_{j=0}^{m-1} y_{2j} e^{\frac{ik\pi j}{\frac{m}{2}}}, \quad O_k = \sum_{j=0}^{m-1} y_{2j+1} e^{\frac{ik\pi j}{\frac{m}{2}}}$$

Then

$$c_k = E_k + e^{\frac{ik\pi}{m}} O_k, \quad c_{k+m} = E_k - e^{\frac{ik\pi}{m}} O_k$$

for $k = 0, 1, 2, ..., m-1$. Hence, the $2m$-point DFT decomposes into two m-point DFTs (of even and odd data).

At the next level, each m-point transformation E_k and O_k can again be split into two $\frac{m}{2}$-point transforms, and so on. After $\log_2(m)$ levels, we reach transformations of size 2, which are trivial to compute.

This recursive divide-and-conquer procedure leads to the computational complexity

$$T(m) = 2T\left(\frac{m}{2}\right) + O(m)$$

which solves to $T(m) = O(m \log_2(m))$.

Algorithm Summary Given $N = 2m$ equally spaced data values $y_0, y_1, ..., y_{N-1}$ (usually $N = 2^p$),

1. **Bit-reversal permutation:** Reorder the input array by reversing binary indices of each data position (this arranges data for in-place computation).

2. **Butterfly operation:** For each stage $r = 1, 2..., \log_2(N)$, compute

$$y_p \leftarrow a + w_r b, \quad y_q \leftarrow a - w_r b$$

where $a = y_p$, $b = y_q$, and $w_r = e^{\frac{i2\pi k}{2^r}}$ are the "twiddle factors".

3. **Output:** The resulting N complex numbers c_k are the Fourier coefficients.

Example 4.3.5. We take $m = 4$, so that there are $N = 2m = 8$ data points. The data are

$$y_j = \sin\left(\frac{\pi j}{4}\right)$$

for $j = 1, 2, ..., 7$. Computing each y_j,

$$\begin{aligned} y_0 &= 0, & y_1 &= \tfrac{\sqrt{2}}{2}, & y_2 &= 1, & y_3 &= \tfrac{\sqrt{2}}{2}, \\ y_4 &= 0, & y_5 &= -\tfrac{\sqrt{2}}{2}, & y_6 &= -1, & y_7 &= -\tfrac{\sqrt{2}}{2} \end{aligned}$$

So setting $\vec{y} = (y_0, y_1, ..., y_7)$, we have

$$\vec{y} = \left(0, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 0, -\frac{\sqrt{2}}{2}, -1, -\frac{\sqrt{2}}{2}\right)$$

We want the length-8 DFT

$$c_k = \sum_{j=0}^{7} y_j e^{\frac{ik\pi j}{4}}, \quad k = 0, 1, 2, ..., 7$$

Define the even and odd subsequences, i.e. for the even indices we have y_0, y_2, y_4, y_6, and for the odd indices we have y_1, y_3, y_5, y_7. For $k = 0, 1, 2, 3$ (i.e. up to $m - 1$), define

$$\begin{aligned} E_k &= y_0 + y_2 e^{\frac{ik\pi}{2}} + y_4 e^{ik\pi} + y_6 e^{\frac{i3k\pi}{2}} \\ O_k &= y_1 + y_3 e^{\frac{ik\pi}{2}} + y_5 e^{ik\pi} + y_7 e^{\frac{i3k\pi}{2}} \end{aligned}$$

Then the FFT combination step is

$$c_k = E_k + e^{\frac{ik\pi}{4}} O_k, \quad c_{k+4} = E_k - e^{\frac{ik\pi}{4}} O_k, \quad k = 0, 1, 2, 3$$

We compute each E_k and O_k as follows. For $k = 0$, we have

$$E_0 = 0, \quad O_0 = 0$$

and so

$$c_0 = 0, \quad c_4 = 0$$

For $k = 1$, we have

$$E_1 = 2i, \quad O_1 = \sqrt{2}(1 + i)$$

and so

$$c_1 = 4i, \quad c_5 = 0$$

For $k = 2$, we have

$$E_2 = 0, \quad O_2 = 0$$

and so

$$c_2 = 0, \quad c_6 = 0$$

Finally, for $k = 3$, we have

$$E_3 = -2i, \quad O_3 = \sqrt{2}(1 - i)$$

and so

$$c_3 = 0, \quad c_7 = -4i$$

Collecting everything

$$\begin{aligned} &c_0 = 0, \quad c_1 = 4i, \quad c_2 = 0, \quad c_3 = 0 \\ &c_4 = 0, \quad c_5 = 0, \quad c_6 = 0, \quad c_7 = -4i \end{aligned}$$

Chapter 5

Numerical Methods for BV Problems of ODEs

Many problems in physics, engineering, and applied mathematics are modelled not by initial value problems (IVPs), but by boundary value problems (BVPs). In a BVP, the solution of a differential equation must satisfy conditions at multiple points (typically at the endpoints of an interval). Classical examples include the equilibrium shape of a beam under load, the temperature profile along a rod, electrolastic potential distributions, and many steady-state models of physical systems.

From a numerical standpoint, BVPs present fundamentally different challenges compared to IVPs. Whereas an IVP evolves a solution forward in one direction from known initial conditions, a BVP must reconcile information from both boundaries, often leading to sensitivity and stability issues. For example, a small error in an initial slope may cause a solution to deviate drastically by the time it reaches the opposite boundary. As a result, numerical schemes for BVPs require different strategies than those used for IVPs.

5.1 The Shooting Method

Boundary value problems (BVPs) for ordinary differential equations arise frequently in mathematical modelling, physics, and engineering. A typical second-order BVP takes the form

$$y'' = f(x, y, y'), \quad a \leq x \leq b, \quad y(a) = \alpha, \quad y(b) = \beta \tag{5.1}$$

Unlike an initial value problem, the solution must satisfy conditions at two distinct points. Direct integration is not possible unless the correct initial slope $y'(a)$ is known ahead of time.

The shooting method resolves this difficulty by transforming the BVP into a sequence of IVPs.

Introduce an unknown initial slope t, and consider the IVP

$$y'' = f(x, y, y'), \quad y(a) = \alpha, \quad y'(a) = t \tag{5.2}$$

For each choice of t, denote the corresponding solution by $y(x,t)$. Our goal is to choose t so that the resulting IVP solution satisfies the boundary value at $x = b$

$$y(b,t) = \beta \tag{5.3}$$

Thus, solving the BVP is equivalent to determining a root of the nonlinear equation

$$\Phi(t) = y(b,t) - \beta = 0 \tag{5.4}$$

A natural idea is to apply Newton's method to the nonlinear scalar equation $\Phi(t) = 0$. Newton's iteration is

$$t_k = t_{k-1} - \frac{\Phi(t_{k-1})}{\Phi'(t_{k-1})} = t_{k-1} - \frac{y(b,t_{k-1}) - \beta}{y'_t(b,t_{k-1})} \tag{5.5}$$

However, computing the derivative

$$y'_t(b,t_{k-1}) = \frac{\partial}{\partial t} y(b,t)\Big|_{t=t_{k-1}}$$

is not feasible analytically for general nonlinear ODEs.

To approximate this derivative, we use the limit definition

$$y'_t(b,t_{k-1}) = \lim_{t \to t_{k-1}} \frac{y(b,t) - y(b,t_{k-1})}{t - t_{k-1}} \approx \frac{y(b,t_{k-2}) - y(b,t_{k-1})}{t_{k-2} - t_{k-1}} \tag{5.6}$$

Substituting (5.6) into (5.5) yields the secant method update formula.

Given two initial guesses t_0 and t_1 and the corresponding endpoint values $y(b,t_0)$ and $y(b,t_1)$, the secant iteration produces

$$t_k = t_{k-1} - (t_{k-1} - t_{k-2}) \frac{y(b,t_{k-1}) - \beta}{y(b,t_{k-1}) - y(b,t_{k-2})} \tag{5.7}$$

for $k \geq 2$. Thus, the nonlinear shooting method proceeds by

1. Choosing initial guesses t_0 and t_1.
2. Solving two IVPs (5.2) with slopes t_0 and t_1.
3. Computing $y(b, t_0)$ and $y(b, t_1)$.
4. Update the slope using (5.7).
5. Repeating until $|y(b, t_k) - \beta|$ is below tolerance.

Each IVP corresponding to a particular guess t must be solved numerically. Let

$$u_1(x) = y(x, t), \quad u_2(x) = y'(x, t)$$

so the second-order ODE becomes a system

$$u_1' = u_2, \quad u_2' = f(x, u_1, u_2), \quad u_1(a) = \alpha, u_2(a) = t \tag{5.8}$$

Let the mesh be

$$h = \frac{b-a}{N}, \quad x_i = a + ih$$

Let $U_{1,i} \approx u_1(x_i)$ and $U_{2,i} \approx u_2(x_i)$. Using Euler's method, for each iteration k and for $i = 0, 1, ..., N-1$,

$$\begin{array}{l} U_{1,i+1}^{(k)} = U_{1,i}^{(k)} + hU_{2,i}^{(k)} \\ U_{2,i+1}^{(k)} + U_{2,i}^{(k)} + hf(x_i, U_{1,i}^{(k)}, U_{2,i}^{(k)}) \end{array}, \quad U_{1,0}^{(k)} = \alpha, \quad U_{2,0}^{(k)} = t_k \tag{5.9}$$

After marching to $x = b$, we read off

$$y(b, t_k) \approx U_{1,N}^{(k)}$$

This value is fed into the secant update (5.7) to compute the next slope.

Example 5.1.1. Consider the BVP

$$y'' = 2y^3, \quad 1 \le x \le 2, \quad y(1) = \frac{1}{4}, \quad y(2) = 1$$

We introduce the unknown initial slope $t = y'(1)$ and consider the IVP

$$y'' = 2y^3, \quad y(1) = \frac{1}{4}, y'(1) = t$$

For each value of t, this IVP has a solution $y(x, t)$. Our goal is to choose t so that

$$y(2, t) = 1$$

Equivalently, we want to solve the scalar equation

$$\Phi(t) = y(2,t) - 1 = 0$$

We need two initial guesses for the slope t_0 and t_1. Let $y_0 = y(x, t_0)$. Then

$$y_0'' = 2y_0^3, \quad 1 \le x \le 2, \quad y_0(1) = \frac{1}{4}, \quad y_0'(1) = t_0$$

We set up the Euler scheme using the standard trick of converting to a first-order system. Define

$$u_1^{(0)} = y_0(x), \quad u_2^{(0)}(x) = y_0'(x)$$

Then

$$(u_1^{(0)})' = u_2^{(0)}, \quad (u_2^{(0)})' = 2(u_1^{(0)})^3$$

On a uniform mesh $x_i = 1 + ih$, $i = 0, 1, 2, ..., N$ with $h = \frac{1}{N}$, we denote

$$U_{1,i}^{(0)} \approx u_1^{(0)}(x_i), \quad U_{2,i}^{(0)} \approx u_2^{(0)}(x_i)$$

Euler's method gives, for $i = 0, 1, 2, ..., N-1$

$$\begin{aligned} U_{1,i+1}^{(0)} &= U_{1,i}^{(0)} + hU_{2,i}^{(0)} \\ U_{2,i+1}^{(0)} &= U_{2,i}^{(0)} + h\left(2(U_{1,i}^{(0)})^3\right) \end{aligned}$$

with initial values $U_{1,0}^{(0)} = \frac{1}{4}$ and $U_{2,0}^{(0)} = t_0$. After marching from $i = 0$ to $i = N$, we obtain

$$U_{1,N}^{(0)} \approx y_0(2) = y(2,0)$$

Similarly, for the second guess t_1, let $y_1(x) = y(x, t_1)$. Then

$$y_1'' = 2y_1^3, \quad 1 \le x \le 2, \quad y_1(1) = \frac{1}{4}, \quad y_1'(1) = t_1$$

Define

$$u_1^{(1)}(x) = y_1(x), \quad u_2^{(1)}(x) = y_1'(x)$$

and approximate on the same mesh

$$U_{1,i}^{(1)} \approx u_1^{(1)}(x_i), \quad U_{2,i}^{(1)} \approx u_2^{(1)}(x_i)$$

Euler's method now gives, for $i = 0, 1, 2, ..., N-1$,

$$\begin{aligned} U_{1,i+1}^{(1)} &= U_{1,i}^{(1)} + hU_{2,i}^{(1)} \\ U_{2,i+1}^{(1)} &= U_{2,i}^{(1)} + h\left(2\left(U_{1,i}^{(1)}\right)^3\right) \end{aligned}$$

with $U_{1,0}^{(1)} = \frac{1}{4}$ and $U_{2,0}^{(1)} = t_1$. After marching to $i = N$, we get

$$U_{1,N}^{(1)} \approx y_1(2) = y(2, t_1)$$

We know the desired boundary condition is $y(2) = 1$, so in terms of the approximations

$$y(2, t_0) \approx U_{1,N}^{(0)}, \quad y(2, t_1) \approx U_{1,N}^{(1)}$$

and we want these to be as close as possible to $\beta = 1$.

The secant method formula (specialized to $\beta = 1$) in the notes is

$$t_k = t_{k-1} - \frac{(t_{k-1} - t_{k-2})(U_{1,N}^{(k-1)} - 1)}{U_{1,N}^{(k-1)} - U_{1,N}^{(k-2)}} \tag{5.10}$$

for $k \geq 2$. Here,

- $U_{1,N}^{(k-1)} \approx y(2, t_{k-1})$,
- $U_{1,N}^{(k-2)} \approx y(2, t_{k-2})$,
- the number $U_{1,N}^{(k-1)} - 1$ is the boundary mismatch at $x = 2$ for the most recent guess,
- the denominator $U_{1,N}^{(k-1)} - U_{1,N}^{(k-2)}$ approximates the change in $y(2, t)$ between the two last slopes.

So we use t_0 and t_1 to compute $U_{1,N}^{(0)}$ and $U_{1,N}^{(1)}$ and we plug it into (5.10) to get t_2.

5.2 Finite Difference Methods

Finite difference methods provide a direct and systematic way to approximate solutions of boundary value problems (BVPs) for ODEs. Unlike the shooting method, which converts a BVP into repeated IVP solves, the finite difference approach discretizes the entire domain and replaces derivatives by algebraic approximations. This leads to a linear (or nonlinear) system that approximates the BVP solution at mesh points.

In this section, we consider the linear second-order BVP

$$Ly = -y''(x) + p(x)y'(x) + q(x)y(x) = g(x), \quad a \leq x \leq b, \quad y(a) = \alpha, \quad y(b) = \beta \tag{5.11}$$

where $p(x)$, $q(x)$, and $g(x)$ are known functions, $q(x) \geq 0$, and $y(x)$ is the unknown solution.

The goal is to construct a discrete approximation $Y_i \approx y(x_i)$ by replacing derivatives with finite differences.

We choose a uniform mesh

$$h = \frac{b-a}{I}, \quad x_i = a + ih, \quad i = 0, 1, 2, ..., I$$

Define the grid function

$$Y_i \approx y(x_i), \quad i = 0, 1, 2, ..., I$$

The boundary condition becomes

$$Y_0 = \alpha, \quad Y_I = \beta$$

We approximate the derivatives in (5.11) using central differences. From Taylor expansion,

$$\frac{y(x_i+h) - y(x_i-h)}{2h} = y'(x_i) + O(h^2)$$

Thus,

$$y'(x_i) \approx \frac{Y_{i+1} - Y_{i-1}}{2h} \tag{5.12}$$

Similarly,

$$y''(x_i) = \frac{y(x_i+h) - 2y(x_i) + y(x_i-h)}{h^2} + O(h^2)$$

Thus, the discrete approximation is

$$y''(x_i) \approx \frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2} \tag{5.13}$$

Both approximations are second-order accurate.

Insert the approximations (5.11) and (5.12) into the differential operator

$$Ly(x_i) = -y''(x_i) + p(x_i)y'(x_i) + q(x_i)y(x_i)$$

We obtain the discrete operator

$$L_h Y_i = -\frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2} + p_i \frac{Y_{i+1} - Y_{i-1}}{2h} + q_i Y_i \tag{5.13}$$

where $p_i = p(x_i)$ and $q_i = q(x_i)$. The finite difference scheme for the BVP is

$$L_h Y_i = g_i, \quad i = 1, 2, ..., I-1 \tag{5.14}$$

with $g_i = g(x_i)$. We have boundary conditions $Y_0 = \alpha$ and $Y_I = \beta$.

Expanding (5.13), we obtain a linear equation for each interior node i

$$\left(-1 - \frac{1}{2}p_i h\right) Y_{i-1} + \left(2 + q_i h^2\right) Y_i + \left(-1 + \frac{1}{2}p_i h\right) Y_{i+1} = g_i h^2 \tag{5.15}$$

This is a three-point stencil involving Y_{i-1}, Y_i, and Y_{i+1}.

For the entire set of interior points $i = 1, 2, ..., I-1$, this produces a tridiagonal linear system

$$A\mathbf{Y} = \mathbf{b} \tag{5.16}$$

where

$$\mathbf{Y} = (Y_1, Y_2, ..., Y_{I-1})^T$$

The matrix A is tridiagonal

$$A = \begin{bmatrix} 2 + q_1 h^2 & -1 + \frac{1}{2}p_1 h & & \\ -1 - \frac{1}{2}p_2 h & 2 + q_2 h^2 & -1 + \frac{1}{2}p_2 h & \\ & \ddots & \ddots & \ddots \\ & & -1 - \frac{1}{2}p_{I-1} h & 2 + q_{I-1} h^2 \end{bmatrix} \tag{5.17}$$

The right-hand side vector $\mathbf{b}$ incorporates $g_i h^2$ and boundary values α and β.

Example 5.2.1. Consider the simple BVP

$$-y'' + y = 1, \quad 0 \le x \le 1, \quad y(0) = 0, \quad y(1) = 1$$

Here,

$$p(x) = 0, \quad q(x) = 1, \quad g(x) = 1$$

Let $h = \frac{1}{5}$ so the mesh points are $x_i = \frac{i}{5}$ for $i = 0, 1, ..., 5$.

The finite difference equations at interior nodes $i = 1, 2, 3, 4$ become

$$-\frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2} + Y_i = 1$$

Multiplying through by h^2,

$$-Y_{i+1} + (2 + h^2)Y_i - Y_{i-1} = h^2$$

Substitute $Y_0 = 0$ and $Y_5 = 1$, yielding the linear system

$$\begin{aligned}(2+h^2)Y_i - Y_2 &= h^2\\ -Y_1 + (2+h^2)Y_2 - Y_3 &= h^2\\ -Y_2 + (2+h^2)Y_3 - Y_4 &+ h^2\\ -Y_3 + (2+h^2)Y_4 &= h^2 + 1\end{aligned}$$

Solving this tridiagonal system gives the approximate values

$$Y_1, Y_2, Y_3, Y_4 \approx y(x_1), y(x_2), y(x_3), y(x_4)$$

This constitutes the finite difference solution.

Remark 5.2.2. The finite difference method produces a sparse, structured linear system (tridiagonal for second-order ODEs). The method is direct (no iteration on slopes as with shooting). It is particularly effective for linear or mildly nonlinear BVPs. It produces second-order accuracy when central differences are used. Solutions can be efficiently computed using the Thomas algorithm (tridiagonal Gaussian elimination).

5.3 Error Analysis

In the previous section, we introduced a second-order, three-point finite difference method for linear boundary value problems (BVPs) of the form

$$Ly \equiv -y'' + p(x)y' + q(x)y = g(x), \quad a \le x \le b$$

with boundary conditions

$$y(a) = \alpha, \quad y(b) = \beta$$

In this section, we analyze the truncation error, stability, convergence, and obtain an error estimate for the finite difference method.

Let the computational mesh be

$$x_i = a + ih, \quad h = \frac{b-a}{I}, \quad i = 0, 1, ..., I$$

The finite difference scheme is

$$L_h Y_i \equiv -\frac{Y_{i-1} - 2Y_i + Y_{i+1}}{h^2} + p_i \frac{Y_{i+1} - Y_{i-1}}{2h} + q_i Y_i = g_i$$

for $i = 1, 2, ..., I-1$ with boundary conditions

$$Y_0 = \alpha, \quad Y_I = \beta$$

Define the truncation error by

$$\tau_i(h) = L_h y_i - [Ly]_i, \quad 1 \le i \le I-1$$

Using Taylor expansions

$$\begin{aligned} y_{i+1} &= y_i + y_i' h + \frac{1}{2} y'' h^2 + \frac{1}{6} y_i^{(3)} h^3 + O(h^4) \\ y_{i-1} &= y_i - y_i' h + \frac{1}{2} y'' h^2 - \frac{1}{6} y_i^{(3)} h^3 + O(h^4) \end{aligned}$$

we obtain

$$\begin{aligned} \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} &= y_i'' + O(h^2), \\ \frac{y_{i+1} - y_{i-1}}{2h} &= y_i' + O(h^2) \end{aligned}$$

Thus,

$$\tau_i(h) = O(h^2), \quad 1 \le i \le I-1$$

and for the boundary conditions

$$\tau_0(h) = \tau_I(h) = 0$$

Therefore, the scheme is second-order consistent.

To analyze stability, consider the model problem

$$-y'' + y = g(x), \quad 0 \le x \le 1$$

with $y(0) = \alpha$ and $y(1) = \beta$. The finite difference scheme becomes

$$-\frac{Y_{i-1} - 2Y_i + Y_{i+1}}{h^2} + Y_i = g_i, \quad 1 \le i \le I-1$$

Multiply by h^2,

$$(2 + h^2) Y_i = Y_{i-1} + Y_{i+1} + g_i h^2$$

Let i_0 satisfy

$$|Y_{i_0}| = \max_{0 \le i \le i} |Y_i|$$

<u>Case 1 (If $1 \le i_0 \le I-1$):</u> Then

$$\begin{aligned}(2+h^2)|Y_{i_0}| &\le |Y_{i_0-1}| + |Y_{i_0+1}| + |g_{i_0}|h^2 \\ &\le 2|Y_{i_0}| + |g_{i_0}|h^2\end{aligned}$$

Subtracting gives

$$h^2|Y_{i_0}| \le |g_{i_0}|h^2$$

hence

$$|Y_{i_0}| \le \max_{1\le i\le I-1} |g_i|$$

<u>Case 2 (If $i_0 = 0$ or $i_0 = I$):</u> Then

$$|Y_{i_0}| = \max\{|\alpha|, |\beta|\}$$

Combining the cases gives the unconditional stability estimate

$$\max_{0\le i\le I} |Y_i| \le \max\{|\alpha|, |\beta|, \max_{1\le i\le I-1} |g_i|\}$$

Let $e_i = y_i - Y_i$. Subtractingt he numerical scheme from the exact discretized equation yields

$$L_h e_i = \tau_i(h), \quad 1 \le i \le I-1$$

with $e_0 = e_I = 0$.

Applying the same stability estimate to the error equation gives

$$\max_{0\le i\le I} |e_i| \le C \max_{1\le i\le I-1} |\tau_i(h)|$$

Since $\tau_i(h) = O(h^2)$, we obtain the global error estimate

$$\max_{0\le i\le I} |y_i - Y_i| = O(h^2)$$

Thus, the method is second-order convergent.