Emory University
MATH 516 Numerical Analysis II
Learning Notes

Jiuru Lyu

June 18, 2025

Contents

List of Algorithms

Numerical Algorithms

What is this course about?

- Nonlinear equations (root finding, fixed point iteration):

  Find $x\ s.t.\ f(x) = 0$, where $f(x)$ is nonlinear.

- Optimization (multivariate):

$$\min_{x \in \mathbb{R}^n} f(x)$$

  First optimality condition: if f is differentiable, $\min_{x \in \mathbb{R}} f(x) \iff f'(x) = 0$

- Interpolation ("connecting the dots"):

  Given (x_i, y_i). Find $f\ s.t.\ y_i = f(x_i)$.

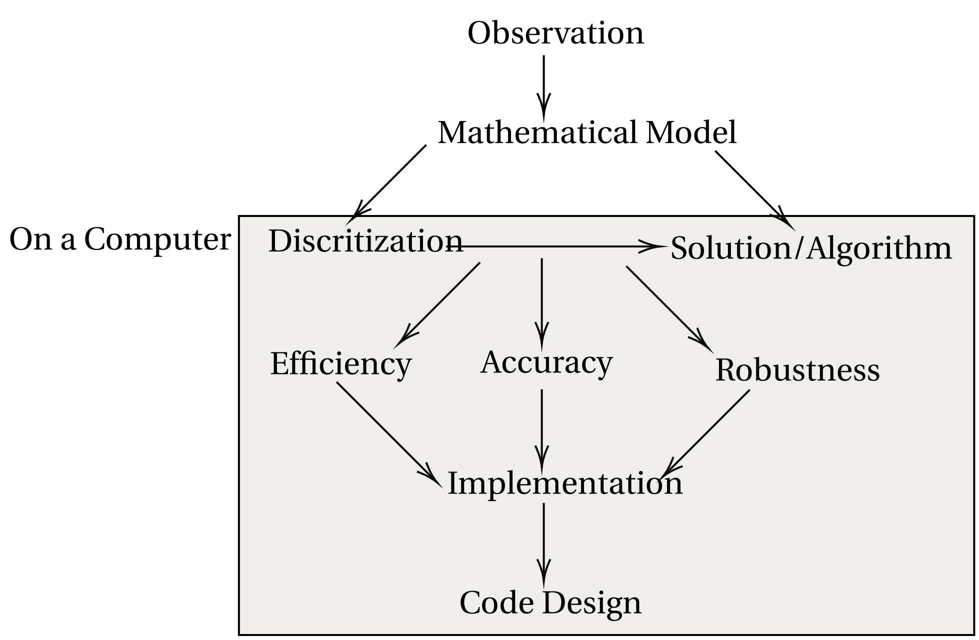
- Differentiation and Integration:

$$f'(x_0) \quad \text{and} \quad \int_a^b f(x)\,\mathrm{d}x$$

- ODEs:

  Solve $y' = f(y, t)$ with $y(t_0) = y_0$.

Scientific Computing



Errors

- Modeling Errors: (often intentional) simplifications of real phenomena to make computation feasible:
  - Approximate of planets as spheres
  - Ignore minor chemical reactions
  - Ignore friction in Physics
  - Approximate a function with (locally) linear models
  - Add regularization
- Approximation errors:
  - Discretize:
  $$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$
  - Convergence: stop early
- Round-off errors:
  - Floating ponts arithmetic
  - Accumulation of error

Big-Oh and Big-Θ Notations

- h: discritization size

$$e(h) = \mathcal{O}(h^q) \iff |e(h)| \leq Ch^q \quad \text{asymptotically as } h \to 0.$$

- n: size of the system/# of points:

$$w(n) = \mathcal{O}(n \log n) \iff |w(n)| \leq Cn \log n \quad \text{as } n \to \infty$$

- $\varphi(n) = \Theta(\psi(n)) \iff c\psi(n) \leq \varphi(n) \leq C\psi(n).$

Accessing an Algorithm

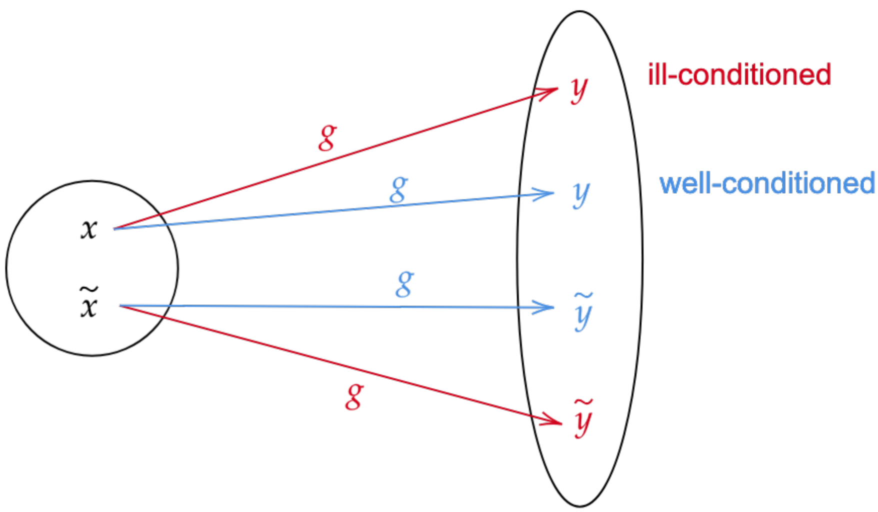
- Accuracy: error, correctness
- Efficiency:

  - flops
  - Rate of convergence
  - Times
  - Parallization/Memory Requirements/... (HPC things)

- Robustness: stability

Problem Conditioning

Let g be the problem:



A stable algorithm yields an exact solution to a nearby problem.

$$\substack{\text{stable}\\\text{algorithm}} + \substack{\text{well-conditioned}\\\text{problem}} = \substack{\text{accurate}\\\text{computed solution}}.$$

Some useful Calculus

Definition 0.0.1 (Taylor Series). Assume that $f(x)$ has $k+1$ derivatives in an interval containing x_0 and x_0+h. Then,

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \cdots + \frac{h^k}{k!}f^{(k)}(x_0) + \frac{h^{k+1}}{(k+1)!}f^{(k+1)}(\xi),$$

where $\xi = \xi_{x_0,h}$ is some point between x_0 and x_0+h.

Remark 1. (Taylor Approximation).

$$f(x_0+h) \approx f(x_0) + h'f(x_0) + \frac{h^2}{2}f''(x_0) + \cdots + \frac{h^k}{k!}f^{(k)}(x_0).$$

Theorem 0.0.2 Intermediate Value Theorem (IVT)
Suppose $f \in \mathcal{C}[a,b]$ and $\widehat{a}, \widehat{b} \in [a,b]$. Let $f(\widehat{a}) \leq s \leq f(\widehat{b})$. Then, $\exists c \in [a,b]\ s.t.\ f(c) = s$.

Theorem 0.0.3 Mean Value Theorem (MVT)
Suppose $f \in \mathcal{C}([a,b])$ and f is differentiable on (a,b). Then, $\exists c \in (a,b)\ s.t.$

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

Theorem 0.0.4 Integral Mean Value Theorem
Suppose $f \in \mathcal{C}([a,b])$, and w is non-negative and integrable on $[a,b]$. That is, $w(x) \geq 0 \quad \forall\, x \in [a,b]$. Then,

$$\int_a^b w(x)f(x)\,\mathrm{d}x = f(\xi)\int_a^b w(x)\,\mathrm{d}x$$

for some $\xi_{a,b} \in [a,b]$.

Remark 2. (Note). Take $w(x) = 1$. Then,

$$\int_a^b f(x)\,\mathrm{d}x = f(\xi)(b-a).$$

By Fundamental Theorem of Calculus,

$$\frac{F(b)-F(a)}{b-a} = f(\xi), \quad \text{where } F(x) \text{ is the antiderivative of } f(x).$$

1 Solving Nonlinear Equations

Goal: Solve $f(x) = 0$ (rooting finding) or solve $g(x) = x$ (fixed point).

- One can convert root finding to fixed point by setting $G(x) = x - f(x)$.
- Alternatively, fixed point problem is equivalent as a root finding problem if one considers $F(x) = g(x) - x$.

Real World Examples:

- Studying planetary motion (Kepler)

$$x = a + b\sin x \qquad \text{(no analytical solution)}$$

- Population growth models:

$$N'(t) = \lambda N(t) + \nu,$$

where λ is the growth rate and ν is the immigration rate. Using ODE techniques, we can solve this equation exactly:

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

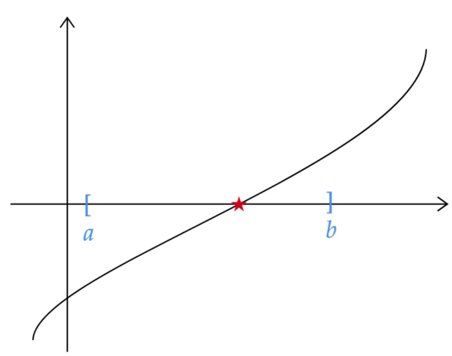
However, if one wants to find growth rate λ^* $s.t.$ $N(1) = 1,000,000$, they need to solve

$$N(1) = N_0 e^{\lambda} + \frac{\nu}{\lambda}\big(e^{\lambda} - 1\big) = 1,000,000.$$

This is a problem with no analytical solution.

1.1 Bisection Method

Goal: Solve $f(x) = 0$ over $[a, b]$. This method is also called the *enclosure method* or the *bracketing method.*



Assumptions:

- $f \in \mathcal{C}([a,b])$
- $f(a)f(b) < 0$: function has different signs at endpoints.

> **Remark.** Why does f have a root in $[a,b]$ under these assumptions? By IVT!

Algorithm 1: Bisection Method

Input: $f \in \mathcal{C}([a,b])$, a, b

begin
    while *not converged* **do**
        compute midpoint $c = \dfrac{a+b}{2}$;
        `// update brackets`
        if $f(b)f(c) \leq 0$ **then**
            $a \leftarrow c$ `// pick the right half`
        else
            $b \leftarrow c$ `// pick the left half`

Output: $\dfrac{a+b}{2}$

---

1. Stopping Criteria:

   - $|f(c)| < \varepsilon$
   - $|b-a| < \varepsilon$
   - number of iterations:

$$N = \left\lceil \log_2 \left(\frac{b-a}{2\varepsilon} \right) \right\rceil .$$

Proof 1. At k-th iteration, the length of the bracket is $(b-a)2^{-k}$. When we stop, we have

$$\frac{(b-a)2^{-k}}{2} < \varepsilon.$$

Then, $|x^* - c_k| < \varepsilon$. Solve for k, we have

$$k > \log_2 \left(\frac{b-a}{2\varepsilon} \right).$$

So we can form a bound for maximum iterations needed to achieve desired level of accuracy. ■

2. Pros and Cons:

   (+) Guaranteed convergence

   (+) Convinient error bound

   (-) Slow

   (-) Can only find simple roots

3. Practical considerations

   - Avoid re-computing f
   - Adjusting tolerance carefully

> **Remark.** In `MATLAB`, `fzero` uses a mix of bisection and interpolation methods.

Example 1.1.1

- Describe the convergence behavior of

$$f(x) = (x-3)^p \quad \text{on } [2,4]$$

for different choices of $p > 0$.

Solution 2.

1. p even: can't use bisection.
2. p odd: convergence $|b-a| < \varepsilon$ will not depend on p. But if we use $|f(c)| < \varepsilon$ as the stopping criteria, the convergence will be dependent on p.

□

- Find a bracket for

$$g(\lambda) = \det(A - \lambda I),$$

where AR is SPD, that it is guaranteed to contain all roots.

Solution 3.

By the Gershgorin disk, we can choose $[0, \star]$, where $g(0) > 0$ (by SPD) and $g(\star) < 0$. One can pick $\star = \|A\|_2^2$ to ensure $g(\star) < 0$. □

1.2 Fixed Point Iteration

Algorithm 2: Fixed Point Iteration

Input: $g \in \mathcal{C}([a, b]])$, initial guess $x_0 \in [a, b]$;

begin
  for $k = 0, 1, \ldots$ **do**
    $x_{k+1} = g(x_k)$;
    if *stopping criteria* **then**
      break;

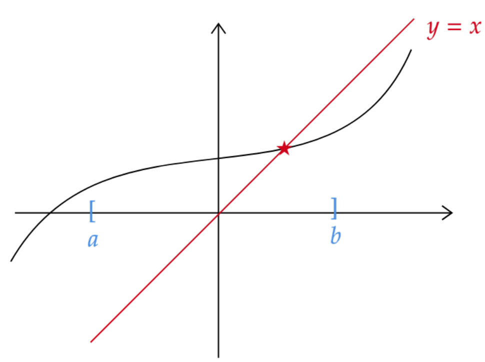
Output: x_{k+1}

Theorem 1.2.1 Fixed Point Theorem/Contraction Mapping Theorem

- Existence: If $g \in \mathcal{C}([a, b])$ with $g(a) \geq a$ and $g(b) \leq b$, then $\exists$ a fixed point $x^* \in [a, b]$.
- Uniqueness: If, in addition, g is Lipschitz continuous with Lipschitz constant ρ and $0 < \rho < 1$:
$$|g(x) - g(y)| \leq \rho|x - y| \quad \forall\, x, y \in [a, b],$$
then the fixed point is unique in $[a, b]$.

Proof 1.

- **Existence**: Define $\varphi(x) = g(x) - x$. Then, $\varphi(a) \geq 0$ and $\varphi(b) \leq 0$. Note that $\varphi(\cdot)$ is continuous. By IVT, $\exists\, x^* \in [a, b]$ $s.t.$ $\varphi(x^*) = 0$. Then, by definition of $\varphi(\cdot)$, $g(x^*) - x^* = 0$, which implies $g(x^*) = x^*$ is a fixed point. □



- **Uniqueness**: Assume $\exists$ another fixed point $y^* \in [a, b]$. Then, by definition of fixed point:
$$|g(x^*) - g(y^*)| \leq |x^* - y^*|.$$

By Lipschitz continuity,

$$|g(x^*) - g(y^*)| \leq \rho |x^* - y^*|.$$

So,

$$|x^* - y^*| \leq \rho |x^* - y^*|.$$

Since $0 < \rho < 1$, we necessarily have $x^* = y^*$. So, the fixed point is unique.

■

Remark 2. (Another Way to Put Uniqueness). If g is differentiable, and $|g'(x)| \leq \rho$ for all x, then we have unique fixed point.

1.2.2 Convergence. Assume g is differentiable and $\rho = |g'(x^*)|$ with $0 < \rho < 1$.

Start with x_0 sufficiently close to x^*, we have

$$\begin{aligned}
x_{k+1} &= g(x_k) \\
\underbrace{x_{k+1} - x^*}_{\text{error}} &= g(x_k) - x^* \\
x_{k+1} - x^* &= g(x_k) - g(x^*) && x^* \text{ is a fixed point} \\
x_{k+1} - x^* &\approx g'(x^*)(x_k - x^*) && \text{MVT} \\
|x_{k+1} - x^*| &\approx \rho |x_k - x^*|.
\end{aligned}$$

So, the error is always decreasing by a factor of ρ.

Example 1.2.3 Another way to Conduct Convergence Analysis

Main Idea: Show error decreases: $e_n = x^* - x_n$.

Assume g is differentiable.

$$\begin{aligned}
e_{n+1} &= x^* - x_{n+1} \\
&= x^* - g(x_n) \\
&= g(x^*) - g(x_n) && x^* \text{ is a fixed point} \\
&= g'(\xi_n)(x^* - x_n) && \text{MVT} \\
e_{n+1} &= g'(\xi_n) e_n.
\end{aligned}$$

When do we converge? $|g'(\xi_n)| < 1 \quad \forall\, n$ eventually.

Definition 1.2.4 (One-sides and Two-sided Convergence). If $g'(x^*) > 0$, then the convergence is *one-sided.* If $g'(x^*) < 0$, then the convergence is *two-sided.*

Example 1.2.5

Consider $g_1(x) = e^{-x}$ and $g_2(x) = -\ln(x)$. Both have a fixed point $x^* \approx 0.56$.

Remark. If g is invertible,

$$fg(x^*) = x^* \implies g^{-1}(x^*) = x^*.$$

- Does FPI with g_1 converge?
  1. $|g_1'(x)| = |e^{-x}| < 1 \implies$ true if $x > 0$. So, we will converge if iterates are positive.
  2. Suppose we started with a bad guess: $x_0 < 0$. Then,

     $$x_1 = g(x_0) = e^{-x_0} > 0.$$

     So, we will always converge, no matter what x_0 we choose.
- Does FPI with g_2 converge?
  1. $|g_2'(x)| = \left|\frac{1}{x}\right| < 1 \implies$ we will converge if $|x| > 1$.
  2. However, $x^* \approx 0.56$ is less than 1. So, we will never converge.

1.3 Newton's Method

Goal: Find root of f: $f(x^*) = 0$.

Assumptions: $f \in \mathcal{C}^2([a, b])$.

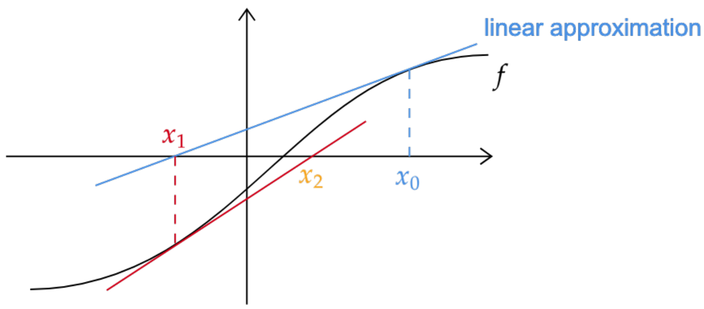
Idea of Newton's Method:

- Consider Taylor Expansion about x_n:

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + f''(\xi_n^{(x)})\frac{(x - x_n)^2}{2}.$$

- Find root of linear approximation:

$$f(x_n) + f'(x_n)(x - x_n) = 0$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$



Remark. It can also be viewed as a fixed point iteration with

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Algorithm 3: Newton's Method

Input: $f \in \mathcal{C}^2([a, b])$, initial guess x_0

begin
  for $k = 0, 1, 2, \ldots$ **do**
    $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$;
    until stopping criteria met.

Output: x_{k+1}

1.3.1 Potential Stopping Criteria.

- Function value:

$$|f(x_k)| < \varepsilon; \quad \frac{|f(x_k)|}{|f(x_0)|} < \varepsilon.$$

- Stagnate:

$$|x_{k+1} - x_k| < \varepsilon$$

- Derivative:

$$|f'(x_k)| < \varepsilon.$$

1.3.2 Pros and Cons.

(+) Fast, converges quadratically (when close to a root)

(+) Local convergence guaranteed

(+) Can find repeated roots

(-) Require smoothness of f

(-) Require derivative evaluations

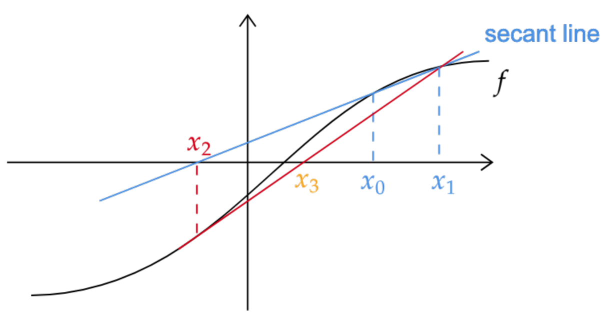
(-) Sensitive to initial guess.

Remark. If we want to relax the requirement of smoothness of f and derivative evaluations while enjoying the fastness of newton's method, then we need to use the *secant method.*

1.4 Secant Method

Main Idea: Newton's method with derivative approximation:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \equiv f[x_{k-1}, x_k] \quad \leftarrow \text{first-order difference}$$



Assumptions:

- $f \in \mathcal{C}^1([a, b])$

- $f(x_0) \neq f(x_1)$.

Algorithm 4: Secant Method

Input: $f \in \mathcal{C}^1([a,b])$, initial guesses x_0, x_1

begin

  for $k = 1, 2, \ldots$ **do**

    $x_{k+1} = x_k - \dfrac{f(x_k)}{f[x_{k-1}, x_k] \approx f'(x_k)} = x_k - \dfrac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$;

    until stopping criteria met

Output: x_{k+1}

1.4.1 Pros and Cons.

(+) Fast, converges superlinearly

(+) Local convergence guaranteed

(+) Only required function evaluations; No need derivative information

(+) Can find repeated roots

(-) Require two initial guesses

(-) Sensitive to initial guesses

1.5 Convergence of Newton's & Secant Methods

Definition 1.5.1 (Rate/Speed of Convergence). Suppose sequence $\{x_n\}_{n=0}^{\infty}$ converges to x^* with $x_n \neq x^* \quad \forall\, n$. We denote this convergence as $x_n \to x^*$. If $\exists\, \lambda \in (0, \infty)$ for $\alpha > 1$ and $\lambda \in (0,1)$ for $\alpha = 1$ $s.t.$

$$\lim_{n\to\infty} \frac{|x^* - x_{n+1}|}{|x^* - x_n|^{\alpha}} = \frac{|e_{n+1}|}{|e_n|^{\alpha}} = \lambda,$$

then $x_n \to x^*$ with order/rate α.

Example 1.5.2 Linearly/Quadratically/Superlinearly Convergent

- Linearly convergent:

$$|x^* - x_{n+1}| \leq \lambda |x^* - x_n|, \quad \lambda \in (0,1).$$

- Quadratically convergent:

$$|x^* - x_{n+1}| \leq \lambda |x^* - x_n|^2$$

- Superlinearly convergent:

$$|x^* - x_{n+1}| \leq \lambda_n |x^* - x_n|,$$

with $\lambda_n \to 0$. For example, $\lambda_n = \lambda |x^* - x_{n-1}|$ or $\lambda_n = \lambda |x^* - x_n|$ (this is actually quadratically convergent! So, quadratically convergent is a special case of superlinearly convergent).

Theorem 1.5.3 Convergence of Newton's Method
Assume $f'(x^*) \neq 0$. Newton's method converges quadratically if x_0 is sufficiently close to x^*.

Proof 1.

- Taylor's series:

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + f''(\xi_n)\frac{(x - x_n)^2}{2}.$$

Replace x with x^*:

$$\begin{aligned}
f(x^*) &= f(x_n) + f'(x_n)(x^* - x_n) + f''(\xi_n)\frac{(x^* - x_n)^2}{2} \\
0 &= f(x_n) + f'(x_n)(x^* - x_n) + f''(\xi_n)\frac{(x^* - x_n)^2}{2} && x^* \text{ is a root} \\
0 &= \frac{f(x_n)}{f'(x_n)} + (x^* - x_n) + \frac{f''(\xi_n)}{f'(x_n)}\frac{(x^* - x_n)^2}{2} && \text{Divide by } f'(x_n) \\
x^* &= \underbrace{\boxed{x_n - \frac{f(x_n)}{f'(x_n)}}}_{x_{n+1}} - \frac{f''(\xi_n)}{f'(x_n)}\frac{(x^* - x_n)^2}{2} \\
x^* - x_{n+1} &= -\frac{f''(\xi_n)}{f'(x_n)}\frac{(x^* - x_n)^2}{2} \\
\implies |x^* - x_{n+1}| &= \frac{|f''(\xi_n)|}{2|f'(x_n)|}|x^* - x_n|^2 && \text{quadratically convergent}
\end{aligned}$$

- What does "sufficiently close" mean?

Let $x_0 \in B_\delta[x^*] \equiv [x^* - \delta, x^* + \delta]$. Choose δ small enough $s.t.$ $f'(x) \neq 0 \quad \forall\, x \in B_\delta[x^*]$. We can do so because f' is continuous. Define

$$M = \frac{\max_{x \in B_\delta[x^*]} |f''(x)|}{2 \min_{x \in B_\delta[x^*]} |f'(x)|} \qquad \text{(the worst case constant)}$$

Then,

$$|x^* - x_{n+1}| \leq M|x^* - x_n|^2.$$

Refining δ: choose δ small enough $s.t.\ M \cdot \delta < 1$.

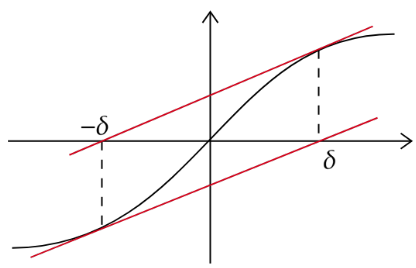
Recall: when we start, $|x^* - x_0| < \delta$. $\implies$ convergence.

■

Example 1.5.4 Importance of Initial Guess

$f(x) = \arctan(x)$ with $x^* = 0$.
Newton's method will converge for any $x_0 \in (x^* - \delta, x^* + \delta)$, for δ small enough.

- What is the largest choice of δ for which we converge?



If $\exists\ \delta\ s.t.$ Newton's method oscillates, this is the largest one.

- How do we find this δ?

$$x_0 = \delta$$
$$x_1 = -\delta = x_0 - \frac{f(x_0)}{f'(x_0)}$$
$$-\delta = \delta - \frac{f(\delta)}{f'(\delta)}$$

Define $h(\delta) = 2\delta - \dfrac{f(\delta)}{f'(\delta)}$. Find the root of $h(\delta) = 0$.
Use Newton's method on h to find δ^*, $\delta^* \approx 1.39$.

2 Optimization

Goal:

$$\min_{x \in \mathbb{R}^n} \varphi(x) \quad \text{where } \varphi(x) : \mathbb{R}^n \to \mathbb{R},\ \varphi \in \mathcal{C}^2.$$

2.1 Multivariable Calculus Review

Definition 2.1.1 (Directional Derivative). If it exists, the *directional derivative* of $\varphi : \mathbb{R}^n \to \mathbb{R}$ at $x \in \mathbb{R}^n$ in direction $d \in \mathbb{R}^n,\ d \neq 0$ is

$$\varphi'(x;\ d) = \lim_{t \to 0} \frac{\varphi(x + td) - \varphi(x)}{t}.$$

Definition 2.1.2 (Partial Derivative). *Partial derivative* is a directional derivative in coordinate direction e_i,

$$\frac{\partial \varphi}{\partial x_i} = \varphi'(x; e_i).$$

Definition 2.1.3 (Gradient). *Gradient*, $\boldsymbol{\nabla}\varphi : \mathbb{R}^n \to \mathbb{R}^n$, is defined as

$$\boldsymbol{\nabla}\varphi = \begin{bmatrix} \frac{\partial \varphi}{\partial x_1} \\ \vdots \\ \frac{\partial \varphi}{\partial x_n} \end{bmatrix}.$$

Lemma 2.4 Directional Derivative:

$$\varphi'(x; d) = \boldsymbol{\nabla}\varphi(x)^\top d,$$

a linear combination of changes in each coordinate.

Theorem 2.1.5 Taylor Series in Several Variables
Given $x \in \mathbb{R}^n$. Assume φ has bounded derivatives up to order at least e. Then, for direction vector $p \in \mathbb{R}^n$, we can write

$$\varphi(x+p) = \varphi(x) + \boldsymbol{\nabla}\varphi(x)^\top p + \frac{1}{2}p^\top \boldsymbol{\nabla}^2\varphi(x)p + \mathcal{O}\big(\|p\|^3\big).$$

Alternatively,

$$\varphi(x+p) = \varphi(x) + \boldsymbol{\nabla}\varphi(x)^\top p + \frac{1}{2}p^\top \boldsymbol{\nabla}^2\varphi(\xi)p, \quad \text{where } \xi \text{ is between } x \text{ and } x+p.$$

Definition 2.1.6 (Hessian/$\boldsymbol{\nabla}^2\varphi(x)$). The *Hessian* of $\varphi(x)$, denoted $\boldsymbol{\nabla}^2\varphi(x)$, is given by

$$\boldsymbol{\nabla}^2\varphi(x) = \begin{bmatrix} \dfrac{\partial^2\varphi(x)}{\partial x_1^2} & \cdots & \dfrac{\partial^2\varphi(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial^2\varphi(x)}{\partial x_n \partial x_1} & \cdots & \dfrac{\partial^2\varphi(x)}{\partial x_n^2} \end{bmatrix} \in \mathbb{R}^{n\times n},$$

where $\left[\boldsymbol{\nabla}^2\varphi(x)\right]_{i,j} = \dfrac{\partial^2\varphi(x)}{\partial x_i \partial x_j}$.

Example 2.1.7

$$p^\top \boldsymbol{\nabla}^2\varphi(x)p = \sum_{j=1}^{n}\sum_{i=1}^{n} \frac{\partial^2\varphi(x)}{\partial x_i \partial x_j} p_i p_j.$$

Definition 2.1.8 (Jacobian). Suppose $F : \mathbb{R}^n \to \mathbb{R}^m$, a vector-valued function,

$$F(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix}, \quad \text{where } f_i : \mathbb{R}^n \to \mathbb{R}.$$

Then, the *gradient* of $F(x)$ is

$$\nabla F(x) = \begin{bmatrix} | & | & & | \\ \nabla f_1(x) & \nabla f_2(x) & \cdots & \nabla f_m(x) \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times m}.$$

The *Jacobian* is

$$J(x) = \nabla F(x)^\top \in \mathbb{R}^{m\times n}.$$

Example 2.1.9 Linear Approximation of $F(x)$

$$F(x+p) \approx F(x) + J(x)p = F(x) + \begin{pmatrix} \nabla f_1(x)^\top p \\ \vdots \\ \nabla f_m(x)^\top p \end{pmatrix}.$$

Remark.

$$\nabla^2 \varphi(x) = \text{Jacobian of } \nabla\varphi \text{ evaluated at } x.$$

Example 2.1.10 Taylor Series for Testing Implementation

We can evaluate φ and $\nabla\varphi$

1. Evaluate at some x:
$$\varphi_0 = \varphi(x) \quad \text{and} \quad g_0 = \nabla\varphi(x).$$
2. Choose search direction $p \neq 0$.
3. Test the linear approximation
$$\begin{aligned} \varphi_1 &= \varphi(x+hp), \quad h \in \mathbb{R} \\ \text{err}_0 &= |\varphi_0 - \varphi_1| \\ \text{err}_1 &= \left|\varphi_0 + hg_0^\top p - \varphi_1\right| \end{aligned}$$

4. Decrease h:

$$\varphi(x+p) = \varphi(x) + \mathcal{O}(h) \qquad \text{(0-th Order Approx.)}$$

$\implies$ cut h in half, err_0 will be cut in half.

$$\varphi(x+p) = \varphi(x) + \nabla\varphi(x)^\top p + \mathcal{O}(h^2) \qquad \text{(1-st Order Approx.)}$$

$\implies$ cut h in half, err_1 will be divided by 4.

Theorem 2.1.11 Optimality Conditions

- First Order (Necessary) Optimality:

  If x^* is a local minimum, then $\nabla\varphi(x^*) = 0$ (or, x^* is a critical point).

- Second Order (Sufficient) Optimality:

  If x^* is a critical point, and

  - $\nabla^2\varphi(x^*) \succ 0$, then x^* is a local minimum;
  - $\nabla^2\varphi(x^*) \prec 0$, then x^* is a local maximum;
  - $\nabla^2\varphi(x^*)$ is indefinite, then x^* is a saddle point.

2.2 Optimization Algorithms

General Algorithm:

$$\min_{x\in\mathbb{R}^n} \varphi(x), \quad \varphi(x) \in \mathcal{C}^2.$$

$$x_{k+1} = x_k + \alpha_k p_k,$$

where α_k is the step size and p_k is the descent direction.

2.2.1 Descent Direction

For a descent direction p, we want $\varphi(x+p) < \varphi(x)$. By Taylor's Series, we have

$$\varphi(x+p) = \varphi(x) + \nabla\varphi(x)^\top p + \mathcal{O}(\|p\|^2).$$

Definition 2.2.1 (Descent Direction). If $\nabla\varphi(x)^\top \neq 0$ and $\|p\|$ is sufficiently small (i.e., we have not met FOC), then a *descent direction* satisfies

$$\nabla\varphi(x)^\top p < 0.$$

Claim 2.2 Suppose $p_k = -B_k^{-1}\nabla\varphi(x_k)$. If B_k is SPD, then p_k is a descent direction.

Proof 1. Since B_k is SPD, B_k^{-1} is also SPD. Then, $y^\top B_k^{-1} y > 0$ if y is nonzero.

$$\begin{aligned}\nabla\varphi(x_k)^\top p_k &= \nabla\varphi(x_k)^\top\left(-B_k^{-1}\nabla\varphi(x_k)\right)\\ &= -\underbrace{\nabla\varphi(x_k)^\top B_k^{-1}\nabla\varphi(x_k)}_{>0} < 0.\end{aligned}$$

■

2.2.3 Ways to Choose B_k.

- $B_k = I$: gradient descent
$$p_k = -\nabla\varphi(x_k).$$
- $B_k = \nabla^2\varphi(x_k)$: Newton's method
$$p_k = -\nabla^2\varphi(x_k)^{-1}\nabla\varphi(x_k).$$
- B_k: secant approximation to Hessian – BFGS (Quasi-Newton's method).

2.2.2 Gradient Descent

Algorithm 5: Gradient Descent (GD)

```
begin
    while not converged do
        p_k = -∇φ(x_k);
```

Pros and Cons:

(+) Simple, only need gradient information

(-) Slow

(-) Sensitive to step size

Remark. One can prove that GD convergence if φ is convex and if $\nabla\varphi$ is Lipschitz continuous (smoothness).

2.2.3 Newton's Method

Algorithm 6: Newton's Method

begin
  while *not converged* **do**
    $p_k = -\nabla^2\varphi(x_k)^{-1}\nabla\varphi(x_k)$;

Proof 2. By FOC, we find the root of $\nabla\varphi(x) = 0$. Build a linear approximation:

$$\nabla\varphi(x+p) \approx \nabla\varphi(x) + \nabla^2\varphi(x)p = 0$$

Then, in each iteration, we need to solve the system

$$\nabla^2\varphi(x)p = -\nabla\varphi(x). \tag{Newton}$$

■

Remark. We can solve (Newton) using Krylov methods, and we don't need to form Hessian explicitly.

Pros and Cons:

(+) Fast, locally quadratic convergence

(+) Scale invariant *(we have the curvature information, so $-\nabla^2\varphi(x)^{-1}$ is rescaling our $\nabla\varphi(x)$ to the right scale. Theoretically, we don't need a line search.)*

(-) Existence of Hessian

(-) Evaluating Hessian is expensive

(-) Solving a linear system at each iteration

(-) Hessian may not be SPD $\implies$ negative curvature (non-descent direction)

2.2.4 BFGS (Quasi-Newton Method)

Definition 2.2.4 (Quasi-Newton Method). The *quasi-Newton method* family approximates the Hessian (so that we don't encounter situations when Hessian does not exist or Hessian is not SPD).

2.2.5 Building up BFGS.

- $x_{k+1} = x_k + p_k$ and $p_k = x_{k+1} - x_k$.
- Taylor's expansion on $\nabla\varphi$:

$$\nabla\varphi(x_{k+1}) \approx \nabla\varphi(x_k) + \nabla^2\varphi(x_k)p_k$$

  We want to estimate the action of Hessian on p_k.
- Iteratively update B_{k+1} to create better and better estimates of $\nabla^2\varphi(x_{k+1})$:
  Assume we already have B_k, and we have computed

$$x_{k+1} = x_k + B_k^{-1}p_k.$$

- We want B_{k+1} to satisfy the secant approximation:

$$B_{k+1}(x_{k+1} - x_k) = \nabla\varphi(x_{k+1}) - \nabla\varphi(x_k).$$

  In 1-D, we have

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

  is an estimation for $\varphi''(\xi)$.
- What properties do we want B_k to satisfy?
  - SPD
  - Easy to solve
  - Easy to update
  - Not too far from B_{k-1}.

2.2.6 Nocedal and Wright Derivation of BFGS.

Main Idea:

$$\min_B \|B - B_k\|_W,$$

such that $B = B^\top$ and $B\underbrace{(x_{k+1} - x_k)}_{p_k} = \underbrace{\nabla\varphi(x_{k+1} - \varphi(x_k)}_{y_k}$.

Definition 2.2.7 (Weighted Frobenius Norm). We choose the *weighted Frobenius norm* as follows:

$$\|A\|_W = \left\|W^{1/2} A W^{1/2}\right\|_F,$$

so that we get unique solution for B and scale invariant rule for W:

$$W \approx -\nabla^2 \varphi(\xi)^{-1} \implies p_k = W y_k.$$

The BFGS choice of W can be derived from MVT

$$\nabla \varphi(x+p) = \nabla \varphi(x) + \int_0^1 \nabla^2 \varphi(x+tp)p \,\mathrm{d}t = \nabla \varphi(x) + \nabla^2 \varphi(\xi) p.$$

Then,

$$W_k = \int_0^1 \nabla^2 \varphi(x_k + t p_k) p_k \,\mathrm{d}t.$$

In this way, W_k captures the curvature information of φ.

2.2.8 Updating B_k**.**

Given B_0, we have

$$B_{k+1} = \left(I - \rho_k y_k p_k^\top\right) B_k \left(I - \rho_k y_k p_k^\top\right)^\top + \rho_k y_k y_k^\top,$$

where

$$\rho_k = \frac{1}{y_k^\top p_k}, \quad \text{and} \quad y_k = \nabla \varphi(x_{k+1}) - \nabla \varphi(x_k)$$

Then, $y_k^\top p_k = (\nabla \varphi(x_{k+1}) - \nabla \varphi(x_k))^\top (x_{k+1} - x_k)$ indicates how much $\nabla \varphi$ changes over the step, and thus is an indication of the curvature information.

Algorithm 7: BFGS, $G_k = B_k^{-1}$

Input: $\varphi, \boldsymbol{\nabla}\varphi, x_0, G_0 = \mu I$

begin

  for $k = 0, 1, \ldots$ **do**

    $p_k = -G_k \boldsymbol{\nabla}\varphi(x_k)$;

    Find step size α_k;

    $x_{k+1} = x_k + \alpha_k p_k$;

    $w_k = \alpha_k p_k$;

    $y_k = \boldsymbol{\nabla}\varphi(x_{k+1}) - \boldsymbol{\nabla}\varphi(x_k)$;

    $\rho_k = \dfrac{1}{y_k^\top w_k}$;

    $G_{k+1} = \left(I - \rho_k w_k y_k^\top\right)^\top G_k \left(I - \rho_k w_k y_k^\top\right)^\top + \rho_k w_k w_k^\top$;

Output: x_{k+1}

2.2.5 Step Size

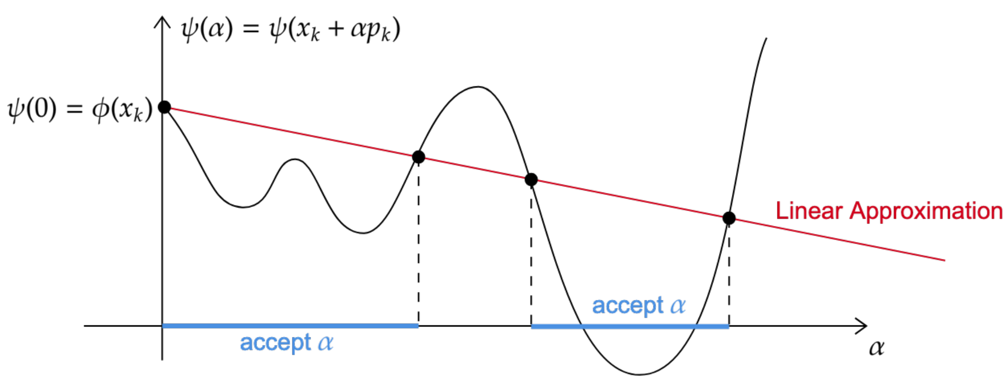
Goal: Choose α $s.t.$

$$\varphi(x_k + \alpha p_k) < \varphi(x_k).$$

We need to satisfy:

- Sufficient decrease condition (Armijo Condition):

$$\varphi(\underbrace{x_k + \alpha p_k}_{x_{k+1}}) \leq \underbrace{\varphi(x_k) + c_1 \alpha \boldsymbol{\nabla}\varphi(x_k)^\top p_k}_{\text{linear approximation}}, \quad c_1 \in (0, 1).$$



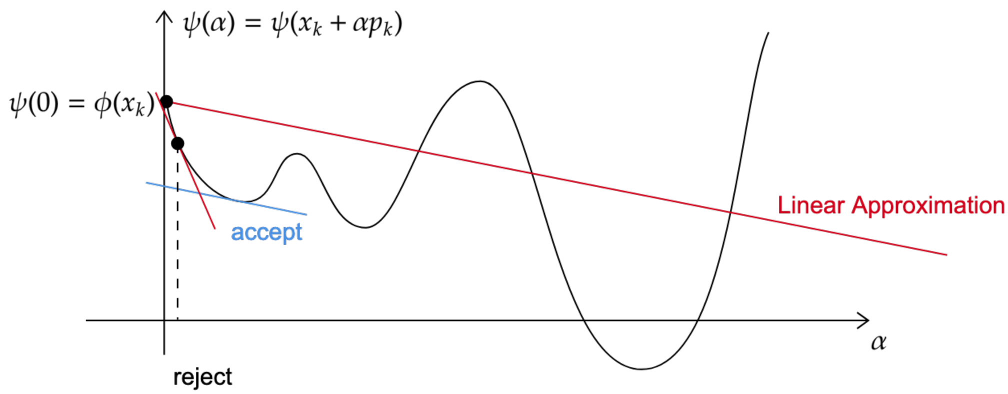
Usually, we take c_1 very small: $c_1 = 10^{-4}$.

Remark. If c_1 is small, we accept more α. If c_1 is large, we reject more α.

Problem: we can take tiny step sizes $\implies$ We need a second condition to avoid so.

- Curvature condition (Wolfe Condition):

$$\underbrace{\nabla\varphi(x+\alpha p_k)^\top p_k}_{\psi'(\alpha)} \geq c_2 \underbrace{\nabla\varphi(x_k)^\top p_k}_{\text{slope of linear approximation}}, \quad 0 < c_1 < c_2 < 1$$



Usually, we take c_2 close to 1: $c_2 = 0.9$.

Algorithm 8: Backtracking Line Search

Input: x_k, p_k, φ, $\nabla\varphi$

begin
- $\widetilde{\alpha}_k = 1$;
- **while** *true* **do**
  - **if** $\varphi(x_k + \widetilde{\alpha}_k p_k) \leq \varphi(x_k) + c_1\widetilde{\alpha}_k \nabla\varphi(x_k)^\top p_k$ *and* $\nabla\varphi(x_k + \widetilde{\alpha}_k p_k)^\top p_k \geq c_2 \nabla\varphi(x_k)^\top p_k$ **then**
    - $\alpha_k = \widetilde{\alpha}_k$;
    - Break;
  - **else**
    - Set $\widetilde{\alpha}_k = \widetilde{\alpha}_k/2$;

2.3 Nonlinear Least Squares and Gauss-Newton

Set-up:

$$\min_{x\in\mathbb{R}^n} \varphi_{\text{LS}}(x) \equiv \frac{1}{2}\|g(x) - b\|_2^2, \quad \text{where } g : \mathbb{R}^n \to \mathbb{R}^m \tag{NLS}$$

- Linear Least Square: $g(x) = Ax$
- General case:

$$\nabla \varphi_{\text{LS}}(x) = \nabla g(x)(g(x) - b) \implies \nabla^2 \varphi_{\text{LS}}(x) = \nabla g(x) \nabla g(x)^\top + L(x)$$

- But what is L? Let's rewrite $\nabla \varphi_{\text{LS}}$ element-wise:

$$\nabla \varphi_{\text{LS}}(x) = \sum_{j=1}^{m} \nabla g_j(x) r_j(x), \quad \text{where } r(x) = g(x) - b.$$

Then,

$$\nabla^2 \varphi_{\text{LS}}(x) = \nabla g(x) \nabla g(x)^\top + \underbrace{\sum_{j=1}^{m} \nabla g_j^2(x) r_j(x)}_{L(x)}.$$

We can view the $L(x)$ as the messy part of Hessian.

2.3.1 Newton's Method for NLS.

$$p = -\left(\nabla g(x) \nabla g(x)^\top + L(x)\right)^{-1} \nabla \varphi(x).$$

2.3.2 Gauss-Newton: Just use the nice stuff.

$$p = -\left(\nabla g(x) \nabla g(x)^\top\right)^{-1} \nabla \varphi(x),$$

where $\nabla g(x) \nabla g(x)^\top$ is a Hessian approximation.

(+) Hessian approx. is symmetric positive semidefinite $\implies$ guaranteed descent direction.

(+) Only need Jacobians $\nabla g(x)^\top \implies$ only first-order derivatives

(+) Converge fast (like Newton) when residual is small

(-) Slower than Newton.

Remark. If the problem is underdetermined, i.e., $n \gg m$, we will get many 0 eigenvalues for $\nabla g(x) \nabla g(x)^\top$. Then, we can introduce regularization

$$\min_{x \in \mathbb{R}^n} \varphi_{\text{LS}}(x) \equiv \frac{1}{2} \|g(x) - b\|_2^2 + \frac{\lambda}{2} \|x\|_2^2,$$

and Gauss Newton becomes $p = -\left(\nabla g(x) \nabla g(x)^\top + \lambda I\right)^{-1} \nabla \varphi(x)$, where $\nabla g(x) \nabla g(x)^\top + \lambda I$ is SPD.

3 Polynomial Interpolation

Goal: Given data points $\{x_i, f(x_i)\}_{i=0}^{n}$ ($n+1$ data points and f is unknown). We want to find a polynomial of degree less than or equal to n, p_n, $s.t.\ p_n(x_i) = f(x_i), \quad i = 0, 1, \ldots, n$.

Procedure:

- Collect the data
- Choose a linearly independent polynomial basis $\{\varphi_0, \varphi_1, \ldots, \varphi_n\}$, where φ_i is a polynomial of degree $\leq n$.
- Construct p_n by fining coefficients $c_0, \ldots, c_n$ $s.t.$

$$p_n(x) = \sum_{j=0}^{n} c_j \varphi_j(x) \quad \text{and} \quad \underbrace{p_n(x_i) = f(x_i), \quad i = 0, \ldots, n}_{\text{interpolation condition}}$$

  To do so, solve a linear system:

$$\begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \varphi_2(x_0) & \cdots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x_n) & \varphi_1(x_n) & \varphi_2(x_n) & \cdots & \varphi_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

- Evaluate p_n at any point x.

Theorem 3.0.1 Uniqueness of Interpolants
For real data points $\{(x_i, y_i)\}_{i=0}^{n}$ with distinct abscissa x_i, $\exists$ a unique polynomial of degree at most n, p_n, which interpolates the data.

3.1 Basis Selection

3.1.1 Monomials.

- Basis: $\{1, x, x^2, \ldots, x^n\}$.
- Construct Coefficients: Vandermonde matrix and solve:

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

- One can show:
$$\det(X) = \prod_{i=0}^{n-1} \left[\prod_{j=i+1}^{n} (x_j - x_i) \right].$$
When is $\det(X) = 0$? When $\exists\, j \neq i$ $s.t.$ $x_j = x_i$. i.e., when x_i's are not distinct.

- Pros and Cons:

  (+) Simple and intuitive

  (+) Evaluate is cheap in nested form (Horner's form): $\sim \mathcal{O}(2n)$. For example, $3x^2 + 2x + 1 = x(3x + 2) + 1$. In each layer, we only need 2 operations.

  (-) Coefficients are hard to interpret

  (-) Have to resolve with slight modification of data points

  (-) Construction is expensive: $\sim \mathcal{O}\left(\frac{2}{3}n^3\right)$, especially for large n. Think of using Gaussian-Elimination.

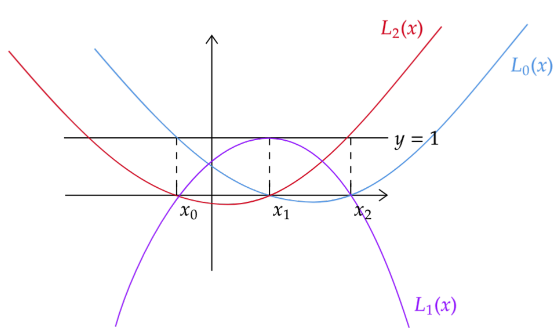
  (-) Vandermonde matrix is often ill-conditioned. (When the interpolation interval is side (round-off or magnitude error) or large n or close x_i's).

3.1.2 Lagrange.

- Basis: $\{L_0(x), L_1(x), \dots, L_n(x)\}$, where
$$L_i(x) = \frac{(x - x_0)(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

- Properties:
  - degree of L_i: n
  - $L_i(x_j) = 0$ for $j \neq i$.
  - $L_i(x_i) = 1$.

- "Standard basis polynomial":



- Construct Coefficients:

$$\begin{bmatrix} L_0(x_0) & L_1(x_0) & \cdots & L_n(x_0) \\ L_0(x_1) & L_1(x_1) & \cdots & L_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ L_0(x_n) & L_1(x_n) & \cdots & L_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$\implies \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}}_{I} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

So,

$$c_i = y_i, \quad \forall\, i = 0, \ldots, n.$$

- The interpolant:

$$p_n(x) = \sum_{i=0}^{n} y_i L_i(x)$$

- Practice Implementation: Barycentric Weights

$$\begin{aligned} \rho_j &= \prod_{i \neq j} (x_j - x_i) \\ &= (x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n) \\ w_j &= \frac{1}{\rho_j}, \quad j = 0, \ldots, n \\ L_j(x) &= w_j \frac{\psi_n(x)}{(x - x_j)}, \quad \text{where} \quad \psi_n(x) = \prod_{i=0}^{n} (x - x_i). \end{aligned}$$

Then,

$$p_n(x) = \psi_n(x) \sum_{j=0}^{n} \frac{w_j y_j}{(x - x_j)}.$$

Imagine $f(x) = 1$, $y_j = 1$, $p_n(x) = 1$ (by uniqueness of interpolants). Then,

$$\begin{aligned} 1 &= \psi_n(x) \sum_{j=0}^{n} \frac{w_j \cdot 1}{(x - x_j)} \\ \psi_n(x) &= \frac{1}{\displaystyle\sum_{j=0}^{n} \frac{w_j}{(x - x_j)}}. \end{aligned}$$

Algorithm 9: Practical Lagrange Interpolation Through Barycentric Weights

Construct barycentric weights w_j and precompute $w_j y_j$ // $\sim \mathcal{O}(n^2)$

Evaluate

$$p_n(x) = \frac{\displaystyle\sum_{j=0}^{n} \frac{w_j y_j}{(x - x_j)}}{\displaystyle\sum_{j=0}^{n} \frac{w_j}{(x - x_j)}} \qquad \text{(Barycentric Interpolation)}$$

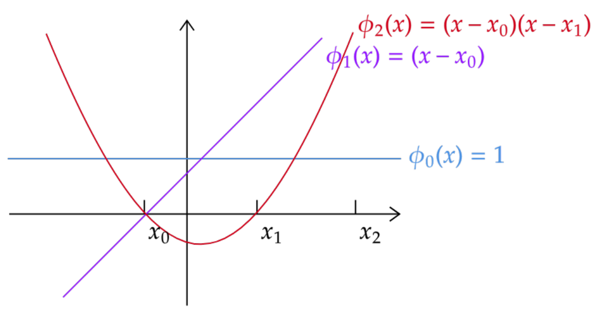
// In numerator and denominator, involves n subtraction, n division, and n summation. So, in total, we have $2 \times 3n = 6n$ operations. Thus, $\sim O(n)$

3.1.3 Newton Polynomials.

- Basis: $\{\varphi_0, \varphi_1, \ldots, \varphi_n\}$, where

$$\varphi_j(x) = \prod_{i=0}^{j-1} (x - x_i).$$

For example, $\varphi_0(x) = 1$, $\varphi_1(x) = (x - x_0)$, and $\varphi_2(x) = (x - x_0)(x - x_1)$



- Constructing Coefficients:

$$\begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \cdots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \cdots & \varphi_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x_n) & \varphi_1(x_n) & \cdots & \varphi_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\Longrightarrow \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & (x_1 - x_0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & \cdots & \displaystyle\prod_{j=1}^{n-1} (x_n - x_j) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \text{lower-triangular system}$$

- Divided Differences:

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \qquad \text{Secant Line}$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \qquad \text{Approximation of second derivative}$$

$$\vdots$$

$$f[x_0, x_1, \ldots, x_k] = \frac{f[x_1, x_2, \ldots, x_k] - f[x_0, x_1, \ldots, x_{k-1}]}{x_k - x_0}$$

- Connecting divided differences with Newton polynomial:

$$\begin{aligned} c_0 &= f[x_0] = f(x_0) \\ c_1 &= f[x_0, x_1] \\ c_2 &= f[x_0, x_1, x_2] \\ &\vdots \\ c_n &= f[x_0, x_1, \ldots, x_n]. \end{aligned}$$

  Specifically, if $0 \leq i \leq j \leq n$:

$$f[x_1, \ldots, x_j] = \frac{f[x_{i+1}, \ldots, x_j] - f[x_i, \ldots, x_{j-1}]}{x_j - x_i}$$

- Then, we can rewrite Newton's polynomial as

$$p_n(x) = \sum_{j=0}^{n} \left[f[x_0, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i) \right]$$

- An analogy to Taylor's approximation:

$$\widetilde{p}_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n.$$

  Newton's polynomial is a secant-like Taylor approximation:

$$p_n(x) = f[x_0] + \underbrace{f[x_0, x_1]}_{\text{secant}}(x - x_0) + \underbrace{f[x_0, x_1, x_2]}_{\text{curvature info}}(x - x_0)(x - x_1) + \cdots + f[x_0, \ldots, x_n](x - x_0)(x - x_{n-1})$$

  When $x_1, \ldots, x_{n-1} \to x_0$*,* $f[x_0, x_1] \to f'(x_0)$ *and* $(x - x_0)(x - x_1) \to (x - x_0)^2$*. Also,* $f[x_0, x_1, x_2] \to f''(x_0)$*, but we differ from Taylor's approximation by the coefficients.*

Table 1: Summary of Bases

Basis	$\varphi_j(x)$	Construction Cost	Evaluation Cost	Pros
Monomial	x^j	$\frac{2}{3}n^3$	$2n$	Simple
Lagrange	$L_j(x)$	n^2	$5n$	$c_j = y_j$; most stable
Newton	$\prod_{i=0}^{j-1}(x-x_i)$	$\frac{3}{2}n^2$	$2n$	Adaptive *(adding new points, no need to reconstruct*

3.2 Error in Polynomial Interpolation

Notation 3.1.

- Divided Differences:

$$f[z_0, z_1, \ldots, z_k] = \frac{f[z_1, \ldots, z_k] - f[z_0, \ldots, z_{k-1}]}{z_k - z_0}$$

- Degree $n+1$ magic polynomial:

$$\begin{aligned}\psi_n(x) &= \prod_{i=0}^{n}(x - x_i) \\ &= (x-x_0)(x-x_1)\cdots(x-x_n)\end{aligned}$$

Theorem 3.2.2 Helper Theorem
Let f be defined and have k bounded derivatives in an interval $[a, b]$. Suppose $z_0, z_1, \ldots, z_k$ be $k+1$ distinct points in $[a, b]$. Then, there is a point $\zeta \in [a, b]$ $s.t.$

$$f[z_0, z_1, \ldots, z_k] = \frac{f^{(k)}(\zeta)}{k!}.$$

Remark 1. (Intuition). Suppose we have z_0 and z_1:

$$\begin{aligned}f[z_0, z_1] &= f'(\zeta) \\ \frac{f(z_1) - f(z_0)}{z_1 - z_0} &= f'(\zeta) && \text{[by MVT!]}\end{aligned}$$

Proof 2. Note that divided differences are invariant to the order of z_i's :

$$f[z_0, z_1, \ldots, z_k] = f[\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_k],$$

where $(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_k)$ is a permutation of $(z_0, z_1, \ldots, z_k)$. *One can prove this claim using induction:*

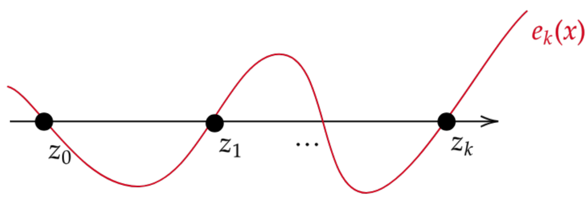
$$f[z_0, z_1] = \frac{f(z_1) - f(z_0)}{z_1 - z_0} = \frac{f(z_0) - f(z_1)}{z_0 - z_1} = f[z_1, z_0].$$

Because we can re-order, we can assume: $a \leq z_0 < z_1 < \cdots < z_k \leq b$. Our approach: construct a Newton interpolant and differentiate. Let p_k be the Newton interpolant with degree at most k. Then,

$$p_k(z_i) = f(z_i) \quad \text{for } i = 0, \ldots, k.$$

Denote the error $e_k(x) = f(x) - p_k(x)$. $\leftarrow$ We will differentiable this!

- Notethat $e_k(z_i) = 0$ as $p_k(z_i) = f(x_i)$



- Note that $p_k(x)$ is of degree at most k:

$$p_k(x) = c_k x^k + q_{k-1}(x)$$

  Then, $p_k^{(k)}(x) = k! c_k = k! f[z_0, z_1, \ldots, z_k]$ WTS: $e_k^{(k)}(x) = f^{(k)}(x) - p_k^{(k)}(x)$ and $\exists\, \zeta \in [a, b]$ $s.t.$ $e_k^{(k)}(\zeta) = 0$. That is,

$$f^{(k)}(\zeta) - k! f[z_0, z_1, \ldots, z_k] = 0.$$

- Scratch: $e_k(z_i)$ has at least $z_0, z_1, \ldots, z_k$ as its roots. So, we have $k - 1$ interval. In each interval, we can apply the Rolle's Theorem to find a x^* $s.t.$ $e^{(1)}(x^*) = 0$. Continuing doing so, we evaluate $e^{(k)}$, and there must be a $\zeta \in (a, b)$ $s.t.$ $e^{(k)}(\zeta) = 0$.

■

Theorem 3.2.3 Error in Polynomial Interpolation
If p_n interpolates f at $n+1$ points $x_0, \ldots, x_n$ and f has $n+1$ bounded derivatives in $[a, b]$, then for each $x \in [a, b]$, $\exists\, \xi = \xi(x) \in [a, b]$ $s.t.$

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \psi_n(x),$$

where $\psi_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$.

Proof 3.

- Error function: $e(x) = f(x) - p_n(x)$. Minimum # of roots of $e(x)$: $n+1$ root at $x_0, \ldots, x_n$. That is, $e(x_i) = 0$ for $i = 0, \ldots, n$.

- Special function:

$$g(x) = e(x) - \frac{\psi_n(x)}{\psi_n(t)} e(t), \quad t \in [a, b]$$

  t is fixed, and we want an expression for $e(t)$ in terms of t. x is the helper variable.

  - $g(x_i) = 0$ for $i = 0, \ldots, n$.

$$g(x_i) = e(x_i) - \frac{\psi_n(x_i)}{\psi_n(t)} e(t) = 0.$$

  - $g(t) = 0$.

$$g(t) = e(t) - \frac{\psi_n(t)}{\psi_n(t)} e(t) = e(t) - e(t) = 0.$$

  - If $t = x_i$, $g(t)$ is not defined, but we define it to be $g(t) = 0$.

$$\lim_{t \to x_i} g(t) = 0.$$

  - If $t \neq x_i$, we have $(n+2)$ roots of g.
  - g is differentiable on (a, b). Composition of differentiable functions: $e(x)$ and $\psi_n(x)$ are differentiable.

- If g has at least $n+2$ roots, then g' has at least $n+1$ roots. Continuing, we know $g^{(n+1)}$ has at least 1 root (repeat Rolle's Theorem). That is, $\exists\, \xi = \xi(t) \in [a, b]$ $s.t.$

$$g^{(n+1)}(\xi(t)) = 0.$$

  Note that

$$g^{(n+1)}(x) = e^{(n+1)}(x) - \frac{\psi_n^{(n+1)}(x)}{\psi_n(t)} e(t).$$

  Since $e(x) = f(x) - p_n(x)$ and $p_n(x)$ has degree at most n,

$$\begin{aligned} e^{(n+1)}(x) &= f^{(n+1)}(x) - \underbrace{p_n^{(n+1)}(x)}_{=0} \\ &= f^{(n+1)}(x). \end{aligned}$$

Since $\psi_n(x) = (x - x_0)(x - x_1)\cdots(x - x_n) = x^{n+1} + q_n(x)$, we know

$$\psi_n^{(n+1)} = (n+1)!$$

So,

$$\begin{aligned} g^{(n+1)}(x) &= e^{(n+1)}(x) - \frac{\psi_n^{(n+1)}(x)}{\psi_n(t)} e(t) \\ &= f^{(n+1)}(x) - \frac{(n+1)!}{\psi_n(t)} e(t). \end{aligned}$$

Plug-in a root $\xi(t)$, we have

$$g^{(n+1)}(\xi(t)) = f^{(n+1)}(\xi(t)) - \frac{(n+1)!}{\psi_n(t)} e(t) = 0.$$

Hence,

$$e(t) = \frac{f^{(n+1)}(\xi(t))}{(n+1)!} \psi_n(t).$$

■

Theorem 3.2.4 Worst Case Error

The worst case error of polynomial interpolation is given by

$$\max_{a \le x \le b} |f(x) - p_n(x)| \le \frac{1}{(n+1)!} \cdot \max_{a \le t \le b} \left| f^{(n+1)}(t) \right| \cdot \max_{a \le s \le b} |\psi_n(s)|.$$

3.3 Chebyshev Interpolation

Can we choose x_i's to get smaller error?

Definition 3.3.1 (Chebyshev Points/Nodes). On interval $[-1, 1]$,

$$x_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right), \quad i = 0, \dots, n.$$

On a general interval $[a, b]$, we apply an affine transformation:

$$x = a + \frac{(b-a)}{2}(t+1), \quad t \in [-1, 1].$$

3.3.2 Goal: Minimize maximum absolute error.

$$\max_{-1\le x\le 1} |f(x) - p_n(x)| \qquad \text{(Worst Case Error)}$$

From Theorem 3.2.4, we know

$$\max_{-1\le x\le 1} |f(x) - p_n(x)| \le \frac{1}{(n+1)!} \underbrace{\boxed{\max_{-1\le z\le 1} \left|f^{(n+1)}(z)\right|}}_{\substack{\text{Hard to predict and}\\ \text{hard to control}}} \cdot \overbrace{\boxed{\max_{-1\le t\le 1} |\psi_n(t)|}}^{\substack{\psi_n(t)=(t-x_0)\cdots(t-x_0)\text{, we}\\ \text{get to choose } x_0 m\ldots,x_n\\ \text{We can control this}}}$$

So, we want to minimize

$$\max_{-1\le x\le 1} |\psi_n(x)| = \max_{-1\le x\le 1} |(x-x_0)(x-x_1)\cdots(x-x_n)|.$$

With Chebyshev points $x_0, \ldots, x_n$,

$$\beta = \min_{x_0,\ldots,x_n} \max_{-1\le x\le 1} |(x-x_0)(x-x_1)\cdots(x-x_n)| = 2^{1-n}.$$

Definition 3.3.3 (Chebyushev Polynomial). On the interval $[-1, 1]$:

- Closed form: $T_n(x) = \cos\left(n\cos^{-1}(x)\right)$
- Recursive form: $T_0(x) = 1$, $T_1(x) = x$, and

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad \text{for } n = 1, 2, \ldots$$

Example 3.3.4 Chebyshev Polynomial

- $T_0(x) = 1$, $T_1(x) = 1\cdot x$
- $T_2(x) = 2xT_1(x) - T_0(x) = 2x^2 - 1$
- $T_3(x) = 2xT_2(x) - T_1(x) = 4x^3 - 3x$
- $T_4(x) = 2xT_3(x) - T_2(x) = 8x^4 - 8x^2 + 1.$

The coefficient of the leading term increase by 2 each time.

Remark 1. (Why Chebyshev Polynomial?).

$$\begin{aligned} T_{n+1}(x) &= \cos\left((n+1)\cos^{-1}(x_i)\right) \quad \leftarrow \text{degree } n+1\text{, has } n+1 \text{ roots.} \\ x_i &= \cos\left(\frac{2i+1}{2(n+1)}\pi\right) \end{aligned}$$

So,

$$\begin{aligned} T_{n+1}(x_i) &= \cos\left((n+1)\cos^{-1}\left(\cos\left(\frac{2i+1}{2(n+1)}\pi\right)\right)\right) \\ &= \cos\left((n+1)\cdot\frac{2i+1}{2(n+1)}\pi\right) \\ &= \cos\left((2i+1)\frac{\pi}{2}\right) \\ &= 0. \end{aligned}$$

Chebyshev points are roots of Chebyshev polynomials.

Then, one can write

$$T_{n+1}(x) = \alpha(x-x_0)(x-x_1)\cdots(x-x_n),$$

where $x_0, x_1, \ldots, x_n$ are Chebyshev points and $\alpha = 2^{n-1}$.

Theorem 3.3.5 Chebyshev Polynomial is the Best
Let p_n be a monic polynomial (leading coefficient $= 1$) of degree n. Then,

$$\max_{-1\leq x\leq 1}|p_n(x)| \geq 2^{1-n}\left(=\frac{1}{2^{n-1}}\right).$$

Remark. We are essentially showing that $\forall\ p_n,\ \max|p_n(x)|$ has a lower bound, and we attempt to show Chebyshev polynomials attain this lower bound. So, we minimize $\max|p_n(x)|$ with Chebyshev polynomials. This only proves existence and we are not showing uniqueness here.

Proof 2. (by contradiction).
Suppose p_n is monic of degree n, and

$$|p_n(x)| < 2^{1-n} \quad \forall\, x\in[-1,1].$$

- Let $q_n(x) = 2^{1-n}T_n(x)$ (normalized Chebyshev polynomial). Note that

$$\max_{-1\le x\le 1} |q_n(x)| = 2^{1-n} \max_{-1\le x\le 1} |T_n(x)| = 2^{1-n}.$$

  Why we normalize Chebyshev polynomial? Because q_n needs to be monic of degree n.

  For $y_i = \cos\left(\frac{i}{n}\pi\right), \quad i = 0,\ldots,n$ we have

$$|q_n(y_i)| = 2^{1-n}.$$

- Look at polynomial $q_n(x) - p_n(x)$, degree $n-1$. Both monic, the n-th degree cancels.
- At y_i's,

$$\underbrace{(-1)^i q_n(y_i)}_{=2^{1-n}} - \underbrace{p_n(y_i)}_{<2^{1-n}} > 0 \qquad T_n(y_i) = \cos(i\pi) = \begin{cases} +1, & i \text{ is even} \\ -1, & i \text{ is odd.} \end{cases}$$

$$(-1)^i[q_n(y_i) - p_n(y_i)] > 0, \quad i = 0,\ldots,n.$$

- $q_n - p_n$ changes signs at least n times in $[-1,1]$.
- $\Longrightarrow$ $q_n - p_n$ has n roots. ※ This contradicts with the fact that $q_n - p_n$ is degree $n-1$.

■

3.4 Interpolation with Derivative Info (Hermite)

Given $t_0,\ldots,t_q$ abscissae and non-negative integers $m_0,\ldots,m_q$.

Goal: Find the unique *osculating* polynomial of lowest degree $s.t.$

$$p_n^{(k)}(t_i) = f^{(k)}(t_i), \quad i = 0,\ldots,q \text{ and } k = m_0,\ldots,m_i.$$

So, each abscissa could have different # of derivative information available.

3.4.1 What is the Minimal Degree n?.

- $m_i = 0$ for $i = 0,\ldots,q$. Only interpolate f, not derivatives

  $\Longrightarrow$ lowest degree $n = q$ (regular old interpolation).

- $q = 0$. Only one abscissa t_0

  $\Longrightarrow$ Taylor approximation of degree m_0.

- $n = 2q + 1$ and $m_i = 1$. Evaluate f and f' at each t_i

  $\implies$ Hermite interpolation

- In general:

$$n = q + \sum_{k=0}^{q} m_k.$$

3.4.2 Hermite Cubic Interpolation.

- We want to construct $p_3(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$.
  - In regular interpolation: use cubic interpolant for 4 abscissae t_0, t_1, t_2, t_3.
  - In Hermite cubic interpolation: only need 2 abscissae t_0 and t_1. $m_0 = 1$ and $m_1 = 1$. Then, $n = q + m_0 + m_1 = 1 + 1 + 1 = 3$ (q counts from 0).
- Finding coefficients:

$$\begin{cases} p_3(t_0) &= f(t_0) \\ p_3(t_1) &= f(t_1) \\ p_3'(t_0) &= f'(t_0) \\ p_3'(t_1) &= f'(t_1) \end{cases} \implies \begin{cases} c_0 + c_1 t_0 + c_2 t_0^2 + c_3 t_0^3 &= f(t_0) \\ c_0 + c_1 t_1 + c_2 t_1^2 + c_3 t_1^3 &= f(t_1) \\ c_1 + 2c_2 t_0 + 3c_3 t_0^2 &= f'(t_0) \\ c_1 + 2c_2 t_1 + 3c_3 t_1^2 &= f'(t_1) \end{cases}$$

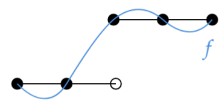
4 Piecewise Interpolation

Previously, we do global interpolant: only one polynomial to connect all dots. Interpolation error is given by

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

(-) Higher order polynomials ten to oscillate

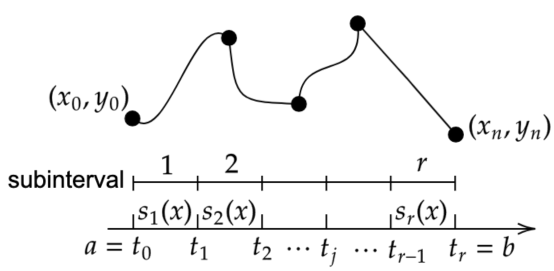
(-) Data may only be piecewise smooth, but polynomial is infinitely smooth.



(-) No locality: changing one data point can drastically change entire interpolant.

4.1 Piecewise Polynomial Interpolation

4.1.1 Overview.



- t_i: break points. From $t_0, \ldots, t_r$, we have $r+1$ break points.
- r: number of subintervals $[t_{i-1}, t_i]$, where $i = 1, \ldots, r$.
- $s_i(x)$: polynomial piece, $i = 1, \ldots, r$.
- $v(x)$: interpolant

$$v(x) = s_i(x) \quad \text{for } t_{i-1} \le x \le t_i, \quad i = 1, \ldots, r.$$

4.1.2 Piecewise Linear.

- Break points: $t_i = x_i$

- Interpolant:
$$v(x) = f(x_{i-1}) + f[x_{i-1}, x_i](x - x_{i-1}), \quad x \in [x_{i-1}, x_i].$$
  (+) Simple
  (+) Max/Min of $v(x)$ are data points $\implies$ No "fake" extrema
  (-) Not differentiable (Give up some smoothness)
  (-) How to extrapolate? (Hard to go beyond the data points)

- **Claim (Error of Piecewise Linear Interpolant)**
$$|f(x) - v(x)| \leq \frac{h^2}{8} \max_{a \leq \xi \leq b} |f''(\xi)|,$$
where $h = \max_{i=1,\ldots,r}(t_i - t_{i-1})$, *max subinterval length.*

Proof 1. On subinterval $[t_{i-1}, t_i]$, we have a linear interpolant. The error is given by
$$f(x) - v(x) = \frac{f''(\xi_i)}{2!}(x - t_{i-1})(x - t_i)$$
Consider $w(x) = (x - t_{i-1})(x - t_i)$. $w(x)$ is minimized at $\frac{t_i + t_{i-1}}{2}$. So,
$$\begin{aligned} |w(x)| = |(x - t_{i-1})(x - t_i)| &\leq \left(\frac{t_i - t_{i+1}}{2}\right)^2 \\ &\leq \frac{h^2}{4}, \end{aligned}$$
where h denotes the largest length of subinterval.

Now, combine everything on interval $[a, b]$:
$$\begin{aligned} |f(x) - v(x)| &\leq \max_{i=1,\ldots,r} \frac{|f''(\xi_i)|}{2} \cdot \frac{h^2}{4} \\ &= \frac{h^2}{8} \max_{a \leq \xi \leq b} |f''(\xi)|. \end{aligned}$$
■

Remark 2. (Implication of This Error Bound). If we double the points, we get quadratic decrease on the error bound.

4.1.3 Piecewise Constant.

- Break points: $t_0 = a$, $t_{i+1} = \frac{x_{i-1} + x_i}{2}$ for $i = 1, \ldots, n$, and $t_{n+1} = b$.

- Interpolant:

$$v(x) = s_i(x) = f(x_{i-1}) \quad t_{i-1} \le x < t_i, \quad i = 1, \ldots, n+1.$$

  (+) Cheap

  (-) No smoothness

- Error bound:

$$|f(x) - v(x)| \le \frac{h}{2} \max_{a \le \xi \le b} |f'(\xi)|.$$

4.1.4 Piecewise Cubic Hermite (Derivative Information).

- Interpolant:

$$v(x) = s_i(x) = a_i + b_i(x - t_{i-1}) + c_i(x - t_{i-1})^2 + d_i(x - t_{i-1})^3, \quad x \in [t_{i-1}, t_i], \quad i = 1, \ldots, r.$$

- Error bound:

$$|f(x) - v(x)| \le \frac{h^4}{4! \cdot 2^4} \max_{a \le \xi \le b} \left| f^{(4)}(\xi) \right| = \frac{h^4}{384} \max_{a \le \xi \le b} \left| f^{(4)}(\xi) \right|.$$

- number of unknowns: $4r$. So, we need $4r$ conditions to solve:

  1. Interpolate condition:

  $$s_i(t_i) = f(t_i)$$

  2. Continuity condition:

  $$s_i(t_i) = s_{i+1}(t_i) = f(t_i)$$

  With 1 and 2, we have $2r$ conditions.

  3. Additional condition: Derivative information:

  $$s_i'(t_i) = s_{i+1}'(t_i) = f'(t_i)$$

  This yields another $2r$ conditions. So, we can solve.

4.2 Cubic Spline Interpolation

4.2.1 What is a Spline?. Consider a spline of order m:

- Knots: $a = x_0 < x_1 < \cdots < x_n = b$
- $v(x)$ is a polynomial of degree $\le m$ on every subinterval $[x_{i-1}, x_i]$.

- $v^{(r)}(x)$ is continuous on (a,b) for $r = 0,\dots,m-1$. That is, $v \in \mathcal{C}^{m-1}[a,b]$.

Example 4.2.2 Cubic Spline

$$s_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3.$$

We impose the following conditions:

- Continuous: $s_i(x_i) = s_{i+1}(x_i)$
- Global smoothness:

$$s_i'(x_i) = s_{i+1}'(x_i) \quad \text{and} \quad s_i''(x_i) = s_{i+1}''(x_i).$$

4.2.3 Cubic Spline Interpolation.

$$s_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, \quad i = 1,\dots,n.$$

- In total, we have $4r$ unknowns.
- Interpolate condition (left endpoint):

$$s_i(x_{i-1}) = f(x_{i-1}). \qquad (r \text{ conditions})$$

- Continuity condition (right endpoint):

$$s_i(x_i) = f(x_i). \qquad (r \text{ conditions})$$

- Additional condition: (global) smoothness at interior points:
  1. First derivative condition:
  $$s_i'(x_i) = s_{i+1}'(x_i) \qquad (r-1 \text{ condition})$$
  2. Second derivative condition:
  $$s_i''(x_i) = s_{i+1}''(x_i) \qquad (r-1 \text{ condition})$$

  Totally, we have $r + r + r - 1 + r - 1 = 4r - 2$ conditions. So, we need 2 more conditions.

- The last two conditions: (Why we need 2 more? We don't have smoothness at endpoints)

1. Free boundary (Natural spline):

$$v''(x_0) = 0 \quad \text{and} \quad v''(x_n) = 0$$

2. Clamped boundary (Complete spline):

$$v'(x_0) = f'(x_0) \quad \text{and} \quad v'(x_n) = f'(x_n).$$

> **Remark.** If we don't have derivative information, this approach does not work. We can also use second order derivative information if we have it.

3. Not-a-knot:

$$s_1'''(x_1) = s_2'''(x_1) \quad \text{and} \quad s_{n-1}'''(x_{n-1}) = s_n'''(x_{n-1}).$$

> **Remark.** This condition makes s_1 and s_2 upto 3 derivatives at x_1. Therefore, the four conditions of s_1 and s_2 match at x_1. Therefore, s_1 and s_2 form a simple cubic, and x_1 is not a knot anymore.

Interpolant	**Local?**	**Order**	**Smooth?**	**Selling features**
Piecewise constant	yes	1	bounded	Accommodates general f
Broken line	yes	2	$\mathcal{C}^0$	Simple, max and min at data values
Piecewise cubic Hermite	yes	4	$\mathcal{C}^1$	Elegant and accurate
Spline (not-a-knot)	not quite	4	$\mathcal{C}^2$	Accurate, smooth, requires only f data

4.3 A Different Perspective on Piecewise Interpolation

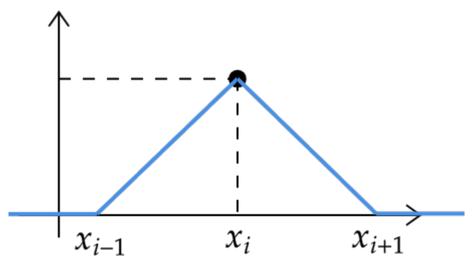
$$v(x) = \sum_{j=0}^{n} c_j \varphi_j(x)$$

Goal: Choose basis functions φ_j that lead to a piecewise approximation. That is, each φ_j has compact support.

4.3.1 Hat Functions (Finite Elements) *Think of Lagrange polynomials*

$$\varphi_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad \text{and } \varphi_j \text{ has compact support,}$$

Having compact support means φ_j is non-zero on a compact set.



$$\varphi_j(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x - x_{i+1}}{x_i - x_{i+1}} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases}$$

To interpolate $(x_i, f(x_i))$,

$$v(x) = \sum_{i=1}^{n} f(x_i)\varphi_i(x).$$

(+) Simple, no need to solve coefficient

(+) Equivalent to linear piecewise interpolation

(-) No smoothness

4.3.2 Hermite Cubic Basis *Adding smoothness*

Goal:

$$v(x) = \sum_{j=0}^{r} \Big[f(x_j) \cdot \xi_j(x) + f'(x_j) \cdot \eta_j(x) \Big] \quad s.t.$$

$$v(x_i) = f(x_i) \quad \text{and} \quad v'(x_i) = f'(x_i) \quad \text{for } i = 0, \ldots, r.$$

Some properties that would be good:

$$\xi_j(x_i) = \delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad \text{and} \quad \eta_j(x_i) = 0$$

$$\xi_j'(x_i) = 0 \quad \text{and} \quad \eta_j'(x_i) = \delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j. \end{cases}$$

To find the basis, let's start on $[0, 1]$:

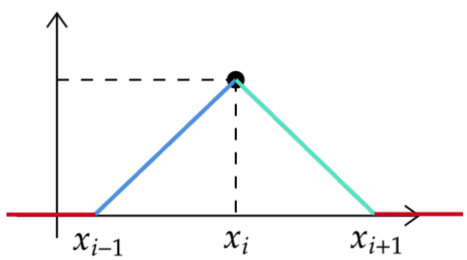
Let $\psi_1, \psi_2, \psi_3, \psi_4$ be cubic polynomials that satisfy:

$$\begin{cases} \psi_1(0) = 1, & \psi_1'(0) = 0, & \psi_1(1) = 0, & \psi_1'(1) = 0 \\ \psi_2(0) = 0, & \psi_2'(0) = 1, & \psi_2(1) = 0, & \psi_2'(1) = 0 \\ \psi_3(0) = 0, & \psi_3'(0) = 0, & \psi_3(1) = 1, & \psi_3'(1) = 0 \\ \psi_4(0) = 0, & \psi_4'(0) = 0, & \psi_4(1) = 0, & \psi_4'(1) = 1 \end{cases}$$

Each $\psi_j(x) = a_i + b_i x + c_i x^2 + d_i x^3 \implies 4$ unknowns. In total, we have 16 unknowns and 16 conditions, so we can solve this system:

$$\implies \begin{cases} \varphi_1(z) = 1 - 3x^2 + 2x^3 \\ \psi_2(z) = z - 2z^2 + z^3 \\ \psi_3(z) = 3z^2 - 2z^3 \\ \psi_4(z) = -z^2 + z^3. \end{cases}$$

Now, extend everything to our original goal:



$$\xi_i(x) = \begin{cases} \psi_3\left(\dfrac{x - x_{i-1}}{x_i - x_{i-1}}\right) & x \in [x_{i-1}, x_i] \\ \psi_1\left(\dfrac{x - x_i}{x_{i+1} - x_i}\right) & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \eta_j = \begin{cases} \psi_4\left(\dfrac{x - x_{i-1}}{x_i - x_{i-1}}\right) & x \in [x_{i-1}, x_i] \\ \psi_2\left(\dfrac{x - x_i}{x_{i+1} - x_i}\right) & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases}$$

5 Best Approximate

5.1 Continuous Least Squares

Recall: Least squares:

$$\min_x \|Ax - b\|_2^2.$$

To solve, we solve a normal equation: $A\top Ax = A^\top b$.

Goal: Approximate a function $f \in \mathcal{F}$ with $v \in \mathcal{F}$ that minimizes

$$\min_{v \in \mathcal{F}} \|f - v\|.$$

5.1.1 Continuous Linear Algebra

- Originally, given $b = Ax$, where $b \in \mathbb{R}^{m\times 1}$, $A \in \mathbb{R}^{m\times n}$, and $x \in \mathbb{R}^{n\times 1}$, we can write

$$b(i) = A(i,:)x = \sum_{j=1}^{n} A(i,j)x(j) \quad \text{for } i = 1, \ldots, m.$$

-

- Suppose $x(j) \in [\ell, u]$ form a uniform discretization. Then,

$$x(j) = \ell + \left(\frac{u-\ell}{n-1}\right)(j-1) \quad \text{for } j = 1, \ldots, n.$$

At the limit $n \to \infty$, we capture the entire interval (continum). So,

$$\lim_{n\to\infty} \sum_{j=1}^{n} A(i,j)x(j) = \int_{\ell}^{u} A(i,x)x \,\mathrm{d}x.$$

We can view b continuously as well:

$$b(y) = \int_{\ell}^{u} A(y,x)x \,\mathrm{d}x \qquad \leftarrow \text{function of } y.$$

In this case, we call $A(y,x)$ a kernel function. To solve for x under this continuous setting, we have

$$x = \int_{\ell}^{u} G(y,x)b(y) \,\mathrm{d}y,$$

where $G(y,x)$ is the Green's function and can be viewed as $x = A^{-1}b$ in the discrete case.

5.1.2 Some Functional Analysis Background

Definition 5.1.1 (Norm). A *norm* for functions on $[a, b]$, $\|\cdot\|$, is a scalar function for all appropriately integrable functions g, f on $[a, b]$ $s.t.$

- $\|g\| \geq 0$ and $\|g\| = 0 \iff g = 0$.
- $\|\alpha g\| = |\alpha| \cdot \|g\| \quad \forall$ scalar α
- $\|g + f\| \leq \|g\| + \|f\|$

Example 5.1.2 Examples of Norms on $[a, b]$

The following norms form functional spaces.

- L_2 norm:

$$\|g\|_2 = \left(\int_a^2 g(x)^2 \, \mathrm{d}x\right)^{1/2} \qquad \text{(least squares)}$$

- L_1 norm:

$$\|g\|_1 = \int_a^b |g(x)| \, \mathrm{d}x$$

- L_∞ norm:

$$\|g\|_\infty = \max_{a \leq x \leq b} |g(x)| \qquad \text{(maximum)}$$

Remark. The higher power we require, we have more regularity on functions (i.e., smoother). So, L_2 is the most restrict one.

Definition 5.1.3 (Orthogonality). Two square-integrable functions, $f, g \in L_2$, are *orthogonal* if $\langle f, g \rangle = 0$, where

$$\langle f, g \rangle = \int_a^b f(x) g(x) \, \mathrm{d}x.$$

5.1.3 Normal Equations of Continuous Least Squares

Goal: Given $f \in L_2$,

$$\min_{v \in V \subset L_2} \|f - v\|_2^2 \quad \rightarrow \text{ infinite dimensional,}$$

where $V = \operatorname{span}\{\varphi_0, \varphi_1, \ldots, \varphi_n\}$ is a subspace of L_2. So,

$$v \in V \iff v(x) = \sum_{j=0}^{n} c_j \varphi_j(x).$$

The optimization problem becomes

$$\min_{c \in \mathbb{R}^{n+1}} \left\| f - \sum_{j=0}^{n} c_j \varphi_j \right\|_2^2 \quad \to \text{ finite dimensional},$$

where $f - \sum_{j=0}^{n} c_j \varphi_j$ is called *residual*, denoted as r.

- Define $\psi_(c) := \left\| f - \sum_{j=0}^{n} c_j \varphi_j \right\|_2^2$ By first order optimality condition: $\nabla \psi(c) = 0$.

$$\begin{aligned}
\frac{\partial \psi}{\partial c_k} &= \frac{\partial}{\partial c_k} \left\| f - \sum_{j=0}^{n} c_j \varphi_j \right\|_2^2 \\
&= \frac{\partial}{\partial c_k} \left[\int_a^b \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right)^2 \mathrm{d}x \right] \\
&= \int_a^b \frac{\partial}{\partial c_k} \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right)^2 \mathrm{d}x \\
&= \int_a^b 2 \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right) (-\varphi_k(x)) \,\mathrm{d}x \\
&= -2 \int_a^b \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right) \varphi_k(x) \,\mathrm{d}x.
\end{aligned}$$

So, by optimality condition, set

$$\frac{\partial \psi}{\partial c_k} = -2 \int_a^b \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right) \varphi_k(x) \,\mathrm{d}x = 0.$$

- Form a linear system to solve for c: Normal Equations

$$\sum_{j=0}^{n} c_j \left[\int_a^b \varphi_j(x) \varphi_k(x) \,\mathrm{d}x \right] = \int_a^b f(x) \varphi_k(x) \,\mathrm{d}x, \quad k = 0, \ldots, n$$
$$\widetilde{B} c = \widetilde{b},$$

where

$$\widetilde{B}_{j,k} = \int_a^b \varphi_j(x)\varphi_k(X)\,\mathrm{d}x = \langle \varphi_j(x), \varphi_k(x) \rangle$$
$$\widetilde{b}_j = \langle f, \varphi_j(x) \rangle .$$

Example 5.1.4

Suppose we are given problem $\|Ax-b\|_2^2$, where $A = \begin{bmatrix} \varphi_0(t) & \varphi_1(t) & \cdots & \varphi_n(t) \end{bmatrix}$. Then, the normal equation is $A \top Ax = A^\top b$, with

$$\widetilde{B}_{j,k} = \left(A^\top A\right)_{j,k} = \varphi_j(t)^\top \varphi_k(t) = \langle \varphi_j, \varphi_k \rangle .$$

- **Claim (Property of $\widetilde{B}$)** *$\widetilde{B}$ is SPD if $\{\varphi_0, \ldots, \varphi_n\}$ is* L.I..
- Residual perspective to solve the system:

$$\frac{\partial \psi(c)}{\partial c_k} = \langle r, \varphi_k \rangle = 0.$$

This implies that residual is orthogonal to basis at the least square solution.

Example 5.1.5 Motivation of Working with Continum

Suppose monomial basis $\varphi_j(x) = x^j$ on $[0,1]$. Then,

$$\widetilde{B}_{j,k} = \langle \varphi_j, \varphi_k \rangle = \int_0^1 x^{j+k}\,\mathrm{d}x = \frac{1}{j+k+1} \quad \text{for } j,k = 0,\ldots,n.$$

So,

$$\widetilde{B}_{j,k} = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & \cdots \\ 1/2 & 1/3 & 1/4 & \ldots & \\ 1/3 & 1/4 & \cdots & & \\ 1/4 & \ldots & & & \\ \vdots & & & & \end{bmatrix} \quad \rightarrow \text{Hilbert matrix; ill-conditioned}$$

Advantage of continuous case: construct better bases.

5.1.6 Two Schools and Thoughts.

- DTO: discretize then optimize.
- OTD: optimize then discretize.

5.1.4 Orthogonal Basis Functions

Goal:

$$\langle \varphi_j, \varphi_k \rangle = 0 \quad j \neq k$$

If we can find such a basis, then $\widetilde{B}$ is diagonal.

Definition 5.1.7 (Legendre Polynomials). On $[-1, 1]$, *Legendre polynomials* are defined recursively as

$$\begin{aligned}
\varphi_0(x) &= 1 \\
\varphi_1(x) &= x \\
\varphi_{j+1}(x) &= \frac{2j+1}{j+1} x \varphi_j(x) - \frac{j}{j+1} \varphi_{j-1}(x), \quad j = 1, 2, \ldots
\end{aligned}$$

Theorem 5.1.8 Properties of Legendre Polynomials

- Orthogonality:

$$\langle \varphi_j, \varphi_k \rangle = \begin{cases} 0, & j \neq k \\ \dfrac{2}{2j+1}, & j = k. \end{cases}$$

  So, the solution to continuous least square is

$$c_j^* = \frac{2j+1}{2} \left[\int_{-1}^{1} f(x) \varphi_j(x) \, \mathrm{d}x \right].$$

  Inverting $\widetilde{B}$ is easy. The work is in computing RHS integrals.

- Calibration: $|\varphi_j(x)| \leq 1$ for $-1 \leq x \leq 1$, and $\varphi_j(1) = 1$.

- Oscillation: φ_j is degree j, and all zeros are simple and lie inside $(-1, 1)$; higher degree, more oscillations.

5.2 Weighted Least Squares

Definition 5.2.1 (Weight Function). A *weight function* is $w : [a, b] \to \mathbb{R}$ $s.t.$

- non-negative: $w(x) \geq 0$, $x \in [a, b]$.
- vanishes ($w(x) = 0$) only at isolated points (a few scattered points in $[a, b]$), if at all.

If $w(x)$ vanishes, it is usually at the endpoints.

Focus: Weighted inner product:

$$\langle f, g \rangle_w = \int_a^b w(x) f(x) g(x) \, \mathrm{d}x \qquad \text{(Intergral mean value theorem)}$$

Proof 1. $\langle f, g \rangle_w$ is a valid inner product:

- positive definiteness: vanishing at isolated points
- symmetry and linearity – as we are integrating.

■

Goal: Find the best approximation $v \approx f$:

$$\min_{v \in V} \langle f - v, f - v \rangle_w \equiv \int_a^b w(x)(f(x) - v(x))^2 \, \mathrm{d}x$$

If $w(x) \equiv 1$, then we are back to the continuous least square setting.

- If $V = \operatorname{span}\{\varphi_0, \ldots, \varphi_n\}$, then $v(x) = \sum_{j=0}^{n} c_j \varphi_j(x)$.

$$\min_{c \in \mathbb{R}^n} \int_a^b w(x) \left(f(x) - \sum_{j=0}^{n} c_j \varphi_j(x) \right)^2 \mathrm{d}x.$$

- Weighted normal equation: $\widetilde{B} c = \widetilde{b}$, where

$$\widetilde{B}_{j,k} = \langle \varphi_j, \varphi_k \rangle_w$$
$$\widetilde{b}_j = \langle \varphi_j, f \rangle_w .$$

We do almost everything the same as before. The only change is that we do a weighted inner product.

- To make $\widetilde{B}$ diagonal, choose orthogonal basis:

$$\langle \varphi_j, \varphi_k \rangle_w = 0 \quad \text{for } j \neq k.$$

  Then,

$$c_j = \frac{\langle \varphi_j, f \rangle_w}{\langle \varphi_j, \varphi_j \rangle_w}.$$

  Solving is cheap. Computing inner products is where the cost comes in.

Question: How does $w(x)$ impact orthogonal basis?

5.2.2 Gram-Schmidt Process to Build an Orthogonal Basis of Functions.

- Recall: Gram-Schmidt process on vectors:

$$\{\vec{\mathbf{a}}_1, \ldots, \vec{\mathbf{a}}_r\} \implies \vec{\mathbf{q}}_j = \vec{\mathbf{a}}_j - \sum_{k=1}^{j-1} \frac{\langle \vec{\mathbf{q}}_k, \vec{\mathbf{a}}_j \rangle}{\langle \vec{\mathbf{q}}_k, \vec{\mathbf{q}}_k \rangle} \vec{\mathbf{q}}_k,$$

  where the inner product for vectors: $\langle \vec{\mathbf{u}}, \vec{\mathbf{v}} \rangle = \vec{\mathbf{u}}^\top \vec{\mathbf{v}}$.

- **Claim (Build an Orthogonal Set of Polynomial based on $\langle \cdot, \cdot \rangle_w$)** *The following procedure works:*

$$\begin{aligned}
\varphi_0(x) &= 1 \\
\varphi_1(x) &= x - \beta_1 \\
\varphi_j(x) &= x\varphi_{j-1}(x) - \beta_j \varphi_{j-1}(x) - \gamma_j \varphi_{j-2}(x) \quad \textit{for } j = 2, 3, \ldots,
\end{aligned}$$

  where

$$\beta_j = \frac{\langle x\varphi_{j-1}, \varphi_{j-1} \rangle_w}{\langle \varphi_{j-1}, \varphi_{j-1} \rangle_w} \quad \textit{for } j = 1, 2, \ldots,$$

  and

$$\gamma_j = \frac{\langle x\varphi_{j-1}, \varphi_{j-2} \rangle_w}{\langle \varphi_{j-2}, \varphi_{j-2} \rangle_w}$$

  Then, $\{\varphi_0, \ldots, \varphi_n\}$ is orthogonal in $\langle \cdot, \cdot \rangle_w$.

  Proof 2. We will prove by induction.

  Base Case

$$\begin{aligned}
\langle \varphi_0, \varphi_1 \rangle_w &= \langle 1, x - \beta_1 \rangle_w \\
&= \langle 1, x \rangle_w - \beta_1 \langle 1, 1 \rangle_w \\
&= \langle 1, x \rangle_w - \langle x, 1 \rangle_w \\
&= 0,
\end{aligned}$$

where $\beta_1 = \dfrac{\langle x\varphi_0, \varphi_0\rangle_w}{\langle \varphi_0, \varphi_0\rangle_w} = \dfrac{\langle x, 1\rangle_w}{\langle 1, 1\rangle_w}$.

Inductive Steps Assume the claim holds for $\{\varphi_0, \dots, \varphi_{j-1}\}$.

Let $\varphi_j(x) = x\varphi_{j-1}(x) - \beta_j\varphi_{j-1}(x) - \gamma_j\varphi_{j-2}(x)$. Then, if $k < j$,

$$\begin{aligned}\langle \varphi_j, \varphi_k\rangle_w &= \langle x\varphi_{j-1} - \beta_j\varphi_{j-1} - \gamma_j\varphi_{j-2}, \varphi_k\rangle_w \\ &= \langle x\varphi_{j-1}, \varphi_k\rangle_w - \beta_j \langle \varphi_{j-1}, \varphi_k\rangle_w - \gamma_j \langle \varphi_{j-2}, \varphi_k\rangle_w \\ &= \langle x\varphi_{j-1}, \varphi_k\rangle_w - \frac{\langle x\varphi_{j-1}, \varphi_{j-1}\rangle_w}{\langle \varphi_{j-1}, \varphi_{j-1}\rangle_w} \langle \varphi_{j-1}, \varphi_k\rangle_w - \frac{\langle x\varphi_{j-1}, \varphi_{j-2}\rangle_w}{\langle \varphi_{j-2}, \varphi_{j-2}\rangle_w} \langle \varphi_{j-2}, \varphi_k\rangle_w\end{aligned}$$

- Case I $k = j-1$. Then, $\gamma_j \langle \varphi_{j-2}, \varphi_k\rangle_w = 0$ by orthogonality. So,

$$\langle \varphi_j, \varphi_k\rangle_w = \langle x\varphi_{j-1}, \varphi_{j-1}\rangle_w - \langle x\varphi_{j-1}, \varphi_{j-1}\rangle_w = 0.$$

- Case II $k = j-2$. Then, $\beta_j \langle \varphi_{j-1}, \varphi_k\rangle_w = 0$ by orthogonality. So,

$$\langle \varphi_j, , \varphi_k\rangle_w = \langle x\varphi_{j-1}, \varphi_{j-2}\rangle_w - \langle x\varphi_{j-1}, \varphi_{j-2}\rangle_w = 0.$$

- Case III $k < j-2$. Then, by orthogonality,

$$\beta_j \langle \varphi_{j-1}, \varphi_k\rangle_w = \gamma_j \langle \varphi_{j-2}, \varphi_k\rangle_w = 0.$$

  Then,

$$\begin{aligned}\langle \varphi_j, \varphi_k\rangle_w &= \langle x\varphi_{j-1}, \varphi_k\rangle_w \\ &= \int_a^b w(x)x\varphi_{j-1}(x)\varphi_k(x)\,\mathrm{d}x \\ &= \int_a^b w(x)\varphi_{j-1}(x)[x\varphi_x(x)]\,\mathrm{d}x \\ &= \langle \varphi_{j-1}, x\varphi_k\rangle_w .\end{aligned}$$

  φ_k is degree-k by construction. So, $x\varphi_k$ has degree $\le j-2$. Then,

$$x\varphi_k = \sum_{i=0}^{j-2} d_i\varphi_i(x).$$

So,

$$\begin{aligned}\langle \varphi_j, \varphi_k \rangle_w &= \left\langle \varphi_{j-1}, \sum_{i=0}^{j-2} d_i \varphi_i \right\rangle_w \\ &= \sum_{i=0}^{j-2} d_i \langle \varphi_{j-1}, \varphi_i \rangle_w \\ &= 0 \quad \text{by orthogonality.}\end{aligned}$$

■

Example 5.2.3 Different Orthogonal Polynomials with Weighted Functions

- Legendre Polynomial: $w(x) \equiv 1, \quad [a,b] = [-1,1]$.

$$\begin{aligned}\varphi_0(x) = 1, \quad \varphi_1(x) = x \\ \varphi_j(x) = \left(\frac{2j+1}{j+1}\right)\varphi_{j-1}(x) - \left(\frac{j}{j+1}\right)\varphi_{j-2}(x).\end{aligned}$$

- Non-compact intervals (Lagaene Polynomial): $w(x) = e^{-x}, \quad [a,b] \to [0,\infty)$.

$$\begin{aligned}\varphi_0(x) = 1, \quad \varphi_1(x) = 1-x \\ \varphi_j(x) = \left(\frac{2j+1-x}{j+1}\right)\varphi_{j-1}(x) - \left(\frac{j}{j+1}\right)\varphi_{j-2}(x).\end{aligned}$$

- Hermite Polynomials (*not the same as Hermite cubic*): $w(x) = e^{-x^2}, \quad [a,b] \to (-\infty,\infty)$.

$$\begin{aligned}\varphi_0(x) = 1, \quad \varphi_1(x) = 2x \\ \varphi_j(x) = 2x\varphi_{j-1}(x) - 2j\varphi_{j-1}(x).\end{aligned}$$

- Chebyshev Polynomials: $w(x) = \dfrac{1}{\sqrt{1-x^2}}, \quad [a,b] = [-1,1]$.

$$\begin{aligned}\varphi_0(x) = 1, \quad \varphi_1(x) = 2x \\ \varphi_j(x) = 2x\varphi_{j-1}(x) - \varphi_{j-2}(x).\end{aligned}$$

6 Numerical Differentiation

6.1 Taylor Series

Definition 6.1.1 (Derivative).

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

Problem in numerical differentiation: we don't know how to evaluate limit. So, we will use *finite differencing* of function evaluations.

General Setting: We can evaluate f but we don't know f' or it is expensive to evaluate f'.

6.1.2 Two-Point Formulas.

- Backward Difference:

$$f(x_0 - h) = f(x_0) - hf'(x_0) + \frac{h^2}{2} f''(\xi) \quad \xi \in [x_0 - h, x_0]$$
$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \underbrace{\frac{h}{2} f''(\xi)}_{\text{truncation error}}.$$

  This method is *first order accurate*: associated truncation error is $\mathcal{O}(h)$. In other words, if h is cut in half, the error is also cut in half.

- Forward Difference

6.1.3 Three-Point Formulas.

- Centered Formula:

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{6} f'''(\xi_1) \quad (1)$$
$$f(x_0 - h) = f(x_0) - hf'(x_0) + \frac{h^2}{2} f''(x_0) - \frac{h^3}{6} f'''(\xi_2) \quad (2)$$

(1) − (2):

$$f(x_0 + h) - f(x_0 - h) = 2hf'(x_0) + \frac{h^3}{6} \underbrace{[f'''(\xi_1) + f'''(\xi_2)]}_{\substack{=2f'''(\xi) \text{ for some} \\ \xi \in [x_0 + h, x_0 - h] \text{ by IVT}}}$$
$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{h^2}{6} f'''(\xi).$$

This method is *second order* accurate: truncation error $\sim \mathcal{O}(h^2)$.

- Higher Order One-Sided Formula:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(\xi_1) \tag{1}$$
$$f(x_0+2h) = f(x_0) + 2hf'(x_0) + \frac{4h^2}{2}f''(x_0) + \frac{8h^3}{6}f'''(\xi_2) \tag{2}$$

$4(1) - (2)$:

$$4f(x_0+h) - f(x_0+2h) = 3f(x_0) + 2hf'(x_0) - \frac{2h^3}{3}f'''(\xi)$$
$$f'(x_0) = \frac{4f(x_0+h) - 3f(x_0) - f(x_0+2h)}{2h} + \frac{h^2}{3}f'''(\xi).$$

This method is also *second order* accurate.

6.1.4 More Points Formula.

- n-points formula: $\sim \mathcal{O}(h^{n-1})$ for odd n.
- We can also try even number of points, but the truncation error can be different.
- We can use Taylor series for higher order derivatives too!

6.2 Interpolate, then Differentiate

Motivation:

- Not all functions are nicely differentiable.
- Taylor series is painful with many points and non-equispaced points.

General Idea: interpolate with Lagrange polynomial, and then differentiate the interpolant.

- $p_n(x) = \sum_{j=0}^{n} f(x_j)L_j(x)$.
- $p'_n(x) = \sum_{j=0}^{n} f(x_j)L'_j(x)$.
- $p'_n(x_0) = \sum_{j=0}^{n} f(x_j)L'_j(x_0)$.

No Matter Which Method We Use, We Will Get the Same Formula.

Example 6.2.1

- Abscissae: x_0, $x_1 = x_0 + h$:

$$p_1(x) = f(x_0) + f[x_0, x_1](x - x_0) \qquad \text{(one-sided formula)}$$

- Interpolation error:

$$f(x) - p_1(x) = (x - x_0)(x - x_1)\frac{f''(\xi)}{2}.$$

As we know that $\frac{f''(\xi)}{2} \equiv f[x_0, x_1, x]$. Then, we have

$$\begin{aligned} f(x) &= p_1(x) + (x - x_0)(x - x_1)f[x_0, x_1, x] \\ f'(x) &= p_1'(x) + \big((x - x_0) + (x - x_1)\big)f[x_0, x_1, x] + (x - x_0)(x - x_1)\frac{\mathrm{d}}{\mathrm{d}x}f[x_0, x_1, x] \\ f'(x_0) &= p_1'(x_0) + (x - x_0)\underbrace{f[x_0, x_1, x_0]}_{=\frac{f''(\xi)}{2}} \end{aligned}$$

7 Numerical Integration

- Basic Quadrature Rules:

$$I(f) = \int_a^b f(x)\,\mathrm{d}x \approx \sum_{j=0}^{n} w_j f(x_j),$$

  where x_j's are abscissae and w_j's are weights.

  - Interpolate, then integrate
  - Newton-Cotes formula (e.g., midpoint, trapezoidal, Simpson's)
  - Stability and DOP.

- Composite Quadrature: integrate in pieces.
- Gaussian Quadrature:
  - Maximize precision by choosing good abscissae.
  - Legender polynomials (orthogonal polynomials).

7.1 Basic Quadrature Rules

7.1.1 $f \approx p_n \implies I(f) \approx I(p_n)$.

- Recall: Lagrange interpolation:

$$p_n(x) = \sum_{j=0}^{n} f(x_j) L_j(x)$$
$$L_j(x) = \prod_{\substack{k=0 \\ k\neq j}}^{n} \frac{(x - x_k)}{(x_j - x_k)}.$$

- Integration:

$$\begin{aligned} I(f) \approx I(p_n) = \int_a^b \sum_{j=0}^{n} f(x_j) L_j(x)\,\mathrm{d}x &= \sum_{j=0}^{n} \int_a^b f(x_j) L_j(x)\,\mathrm{d}x \\ &= \sum_{j=0}^{n} f(x_j) \underbrace{\int_a^b L_j(x)\,\mathrm{d}x}_{w_j} \\ &= \sum_{j=0}^{n} w_j f(x_j). \end{aligned}$$

Example 7.1.2 Trapezoidal Rule

Suppose $n = 1$, $x_0 = a$, and $x_1 = b$. Then,

$$L_0(x) = \frac{x-b}{a-b} \quad \text{and} \quad L_1(x) = \frac{x-a}{b-a}.$$

So,

$$\begin{aligned} w_0 &= \int_a^b L_0(x)\,\mathrm{d}x = \int_a^b \frac{x-b}{a-b}\,\mathrm{d}x = \frac{b-a}{2} \\ w_1 &= \int_a^b L_1(x)\,\mathrm{d}x = \int_a^b \frac{x-a}{b-a}\,\mathrm{d}x = \frac{b-a}{2}. \end{aligned}$$

Then,

$$\begin{aligned} I(f) &\approx \sum_{j=0}^{n} w_j f(x_j) \\ &= \frac{b-a}{2} f(a) + \frac{b-a}{2} f(b) \\ &= \frac{b-a}{2}(f(a) + f(b)). \end{aligned} \qquad \text{(Trapezoidal Rule)}$$

This method uses *linear interpolant* and *abscissae include endpoints*

Theorem 7.1.3 Midpoint Rule

$$I(f) \approx \sum_{j=0}^{n} w_j f(x_j) = (b-a) f\left(\frac{a+b}{2}\right).$$

- Constant interpolant (p_0)
- Abscissae do not include endpoints.

Theorem 7.1.4 Simpson's Rule

$$I(f) \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

- Quadratic interpolant
- Abscissae include endpoints.

Definition 7.1.5 (Newton-Cotes Formula). *Newton-Cotes formulas* refers to the quadrature rules that are based on interpolation with equispaced abscissae.

- Closed: abscissae include endpoints.
- Open: abscissae exclude endpoints.

7.2 Error in Quadrature

$$\begin{aligned} E(f) &= I(f) - \sum_{j=0}^{n} w_j f(x_j) \\ &= I(f) - I(p_n) \\ &= I(f - p_n) && \text{[Integration is linear]} \\ &= \int_a^b f[x_0, \dots, x_n, x]\, \underbrace{(x - x_0)(x - x_1)\cdots(x - x_n)}_{\psi_n(x)}\, \mathrm{d}x && \text{[Interpolation error]} \end{aligned}$$

Example 7.2.1 Error of Trapezoidal Rule

$$\begin{aligned} E(f) &= \int_a^b f[a, b, x]\, \underbrace{(x - a)(x - b)}_{\psi_1(x) \leq 0\ \forall\, x \in [a,b]}\, \mathrm{d}x \\ &= f[a, b, \xi] \int_a^b (x - a)(x - b)\, \mathrm{d}x \quad \text{for some } \xi \in [a, b] && \text{[Integral MVT]} \\ &= \underbrace{f[a, b, \xi]}_{= \frac{f''(\eta)}{2!}} \left(-\frac{(b - a)^3}{6} \right) \\ &= -\frac{f''(\eta)}{12}(b - a)^3. \end{aligned}$$

- The negative sign indicates that if $f''(\eta) > 0$, $E(f) < 0$, we are over estimating the integral. On the other hand, if $f''(\eta) < 0$, $E(f) > 0$, then we are under estimating.
- $(b - a)^3$: If the interval is cut in half, the accuracy will be improved by 8 times.

Theorem 7.2.2 Errors in Quadrature Rules

- Midpoint:

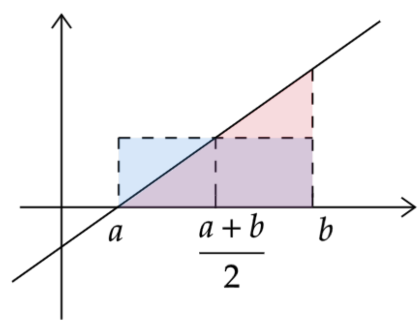
$$E(f) = \frac{f''(\eta)}{24}(b-a)^3$$

- Simpson's:

$$E(f) = -\frac{f^{(4)}(\eta)}{90}\left(\frac{b-a}{2}\right)^5$$

Example 7.2.3 Midpoint Rule is Superconvergence

Note that for midpoint rule: $I(f) = I(p_0)$. So, we don't make mistakes for linear terms and functions. We start to make mistakes for quadratic functions since the second derivative show up in the error term.



Therefore, we are using a degree 0 interpolant to exactly interpolate the integral of degree 1 polynomials. We call this property *superconvergence*.

Definition 7.2.4 (Precision/Degree of Accuracy/Degree of Precision (DOP)). The degree of precision is the largest integer ρ $s.t.$

$$E(q_n) = 0 \quad \forall n \le p,$$

where q_n is a degree-n polynomial.
In other words, we have $I(q_n) - I(p_k) = 0$, where p_k is a degree-k interpolant of q_n.

Theorem 7.2.5 Precision of Quadrature Rules

- Trapezoidal Rule: $\rho = 1$;
- Midpoint Rule: $\rho = 1$; and
- Simpson's Rule: $\rho = 3$.

The midpoint rule and Simpson's rule have superconvergence.

7.3 Composite Quadrature Rules

$$\begin{aligned} I(f) &= \int_a^b f(x)\,\mathrm{d}x \\ &= \sum_{i=1}^{r} \int_{t_{i-1}}^{t_i} f(x)\,\mathrm{d}x \\ &\approx \sum_{i=1}^{r} \underbrace{\int_{t_{i-1}}^{t_i} p^i(x)\,\mathrm{d}x}_{\text{some quadrature}} \end{aligned}$$

7.4 Gaussian Quadrature

Goal: Maximize precision by choosing the right abscissae.

$$I(f) \approx \sum_{j=0}^{n} w_j f(x_j).$$

$n+1$	abscissae	x_j
$n+1$	weights	w_j
$2n+2$ degree of freedom		
$\implies$ exactly integrate degree $(2n+1)$ polynomial		

This degree-$(2n+1)$ polynomial is our target max precision.

7.4.1 Error and Precision.

- A quadrature rule has DoP$= m$ if

$$E(q_k) = \int_a^b q_k(x)\,\mathrm{d}x - \sum_{i=0}^{n} w_i q_k(x_i) = 0$$

for $k = 0, \ldots, m$, where q_k is a degree-k polynomial.

- So,

$$E(f) = \int_a^b [f(x) - p_n(x)]\,\mathrm{d}x = \int_a^b f[x_0, x_1, \ldots, x_n, x] \underbrace{\prod_{i=0}^{n}(x - x_i)}_{\substack{\text{degree } n+1 \\ \varphi_{n+1}(x) \text{ Legendre poly.}}} \mathrm{d}x$$

- We will choose abscissae to be the roots of Legendre polynomial $\varphi_{n+1}(x)$.
- Observation: Suppose $f[x_0, x_1, \ldots, x_n, x]$ is a polynomial of degree n or less.

  Then, $E(f) = 0$.

  Proof 1.

$$f[x_0, x_1, \ldots, x_n, x] = \sum_{k=0}^{n} c_k \varphi_k(x).$$

$$E(f) = \sum_{k=0}^{n} c_k \int_{-1}^{1} \underbrace{\varphi_k(x)\varphi_{n+1}(x)}_{\text{orthogonal}}\,\mathrm{d}x = 0$$

■

- If f is a polynomial, what degree will ensure $f[x_0, x_1, \ldots, x_n, x]$ is degree n?

  Solution 2.

$$\underbrace{f[x_0, x_1, \ldots, x_n, x]}_{\text{degree } n} = \frac{\overbrace{f[x_1, \ldots, x_n, x]}^{\text{degree } n+1} - f[x_0, \ldots, x_n]}{(x - x_0)}$$

$$= \frac{\dfrac{\overbrace{f[x_2, \ldots, x_n, x]}^{\text{degree } n+2} - c_1}{(x - x_1)} - c_0}{(x - x_0)}$$

$$\vdots$$

$$= \frac{\overbrace{f[x]}^{\text{degree } 2n+1} - \text{constant}}{(x - x_0)(x - x_1)\cdots(x - x_n)}.$$

□

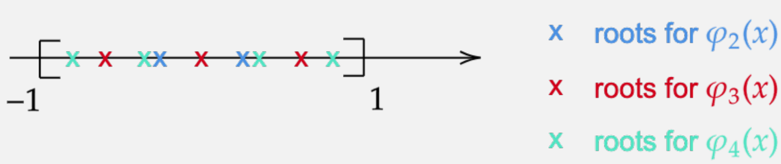
- If we choose $x_0, \ldots, x_n$ to be roots of $\varphi_{n+1}(x)$, then our interpolatory quadrature rule has DoP of $2n + 1$. This way to choose the abscissae is called *Gauss Quadrature.*

Theorem 7.4.2 Properties of Legendre Polynomials

- Orthogonal:

$$\int_{-1}^{1} \varphi_k(x)\varphi_j(x)\,\mathrm{d}x = 0 \quad k \neq j.$$

- $\varphi_j(x)$ is degree-j.
- $\varphi_n(x)$ has n real simple roots in $(-1, 1)$.
- Interlacing property:



Example 7.4.3 Gauss Quadrature

On interval $[-1, 1]$, Legendre polynomials:

$$\varphi_0(x) = 1, \quad \varphi_1(x) = x, \quad \varphi_2(x) = \frac{1}{2}\left(3x^2 - 1\right), \quad \varphi_3(x) = \frac{1}{2}\left(5x^3 - 3x\right).$$

1. $n = 0$: abscissae: x_0, weight w_0.

   - x_0: root of $\varphi_1(x)$: $x_0 = 0$.
   - Target DoP: $2n + 1 = 1$.

$$E(x^0) = \int_{-1}^{1} 1\,\mathrm{d}x - \underbrace{w_0}_{=w_0 f(x_0)} = 0 \implies w_0 = 2.$$

$$E(x^1) = \int_{-1}^{1} x\,\mathrm{d}x - w_0 x_0 = 0 \implies \text{always true}$$

   So, Gauss quadrature with $n = 0$:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx 2f(0).$$

   This is the *midpoint rule*.

2. $n = 1$. Abscissae: x_0 and x_1; weights w_0 and w_1.

- Root of $\varphi_2(x) = \frac{1}{2}(3x^2 - 1) = 0 \implies x_0 = -\frac{\sqrt{3}}{3}, x_1 = \frac{\sqrt{3}}{3}$.
- Target DoP: $2n + 1 = 3$.

$$E(x^0) = \int_{-1}^{1} 1 \,\mathrm{d}x - w_0 x_0^0 - w_1 x_1^0 = 0 \implies w_0 + w_1 = 2$$
$$E(x^1) = \int_{-1}^{1} x \,\mathrm{d}x - w_0 x_0 - w_1 x_1 = 0 \implies -w_0 + w_1 = 0$$
$$E(x^2) = \int_{-1}^{1} x^2 \,\mathrm{d}x - w_0 x_0^2 - w_1 x_1^2 = 0 \implies \frac{1}{3}w_0 + \frac{1}{3}w_1 = \frac{2}{3}$$
$$E(x^3) = \int_{-1}^{1} x^3 \,\mathrm{d}x - w_0 x_0^3 - w_1 x_1^3 = 0 \implies -w_0 + w_1 = 0.$$

We only need to solve

$$\begin{cases} w_0 + w_1 = 2 \\ -w_0 + w_1 = 0 \end{cases} \implies \begin{cases} w_0 = 1 \\ w_1 = 1 \end{cases}$$

So, Gauss quadrature with $n = 1$:

$$\int_{-1}^{1} f(x) \,\mathrm{d}x \approx f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right).$$

- Recall: Simpson's method also have DoP$= 3$. We used quadratic interpolant that requires 3 abscissae. However, with Gauss quadrature, we only need 2 abscissae.

3. Another way to derive Gauss quadrature: solve x_0, x_1, w_0, w_1 from the system.

Theorem 7.4.4 Weights of Gauss Quadrature

$$w_j = \frac{2(1 - x_j)^2}{[(n+1)\varphi_n(x_j)]^2} \quad \text{for } j = 0, \ldots, n.$$

To compute the Gauss Quadrature on $[a, b]$, we consider abscissae $t_j \in [a, b]$. Let $t \in [a, b]$ such that

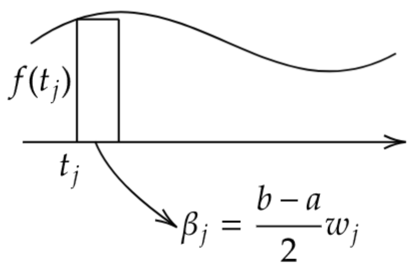
$$t = \left(\frac{b-a}{2}\right)x + \left(\frac{b+a}{2}\right), \quad x \in [-1, 1]$$
$$\mathrm{d}t = \left(\frac{b-a}{2}\right)\mathrm{d}x$$

Then,

$$\begin{aligned}\int_a^b f(t)\,\mathrm{d}t &= \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right)\left(\frac{b-a}{2}\right)\mathrm{d}x \\ &\approx \sum_{j=0}^{n} \beta_j f(t_j),\end{aligned}$$

where $t_j \in [a,b]$ are abscissae such that

$$\begin{aligned} t_j &= \left(\frac{b-a}{2}\right)x_j + \left(\frac{b+a}{2}\right) \\ \beta_j &= \left(\frac{b-a}{2}\right)w_j. \end{aligned}$$



Definition 7.4.5 (Weighted Gauss Quadrature). when computing weighted integrals, we will use weighted Gauss quadrature. Procedure:

- Choose orthogonal basis based on weighted integral.
- Abscissae: roots of $\varphi_{n+1}^{w}(x)$
- General quadrature rule:

$$\int_a^b f(x)w(x)\,\mathrm{d}x = \sum_{j=0}^{n} a_j f(x_j).$$

7.5 Adaptive Quadrature

Main Idea: We will continuing refining the partition on regions where the error is the largest.

Question: How do we compute error?

$$E(f) = E(f;h) = Kh^q + \mathcal{O}\big(h^{q+1}\big), \quad K = \left\|f^{(m)}(\eta)\right\|$$

Let's choose two quadrature rules on each partition. One with step size h and the other with a finer step size $\frac{h}{2}$. Then,

$$E_1(f) = I(f) - R_1 \approx Kh^1$$
$$E_2(f) = I(f) - R_2 \approx K\left(\frac{h}{2}\right)^q \approx \frac{1}{2^q}E_1.$$

Then,

$$R_1 - R_2 \begin{cases} \text{large: we need to refine} \\ \text{small: we are close.} \end{cases}$$

Goal: Choose abscissae as we go such that

$$\underbrace{|I(f) - Q(f; t_0, \ldots, t_r)|}_{\text{error}} < \text{tolerance},$$

where $Q(\cdot)$ is any quadrature rule, and $t_0, \ldots, t_r$ are abscissae.

- Notation: $Q(f; h)$ where $h = \max_{i=1,\ldots,r} t_i - t_{i-1}$.

$$E(f; h) = I(f) - Q(f; h).$$

- Main idea: Use error estimates $E(f; h) = Kh^q + \mathcal{O}(h^{q+1})$, where K depends on f, f', a, and b, but K is independent of h.

Example 7.5.1 Priori Error Estimates

1. Composite trapezoid:

$$E(f; h) \le \underbrace{\frac{\|f''\|_\infty}{12}(b-a)}_{K} h^2$$

2. Composite midpoint:

$$E(f; h) \le \frac{\|f''\|_\infty}{24}(b-a)h^2$$

3. Composite Simpson:

$$E(f; h) \le \frac{\|f^{(4)}\|_\infty}{180}(b-a)h^4.$$

These are called a priori error estimates (before computation). *However, they are not useful in practice because we don't know much about f*

- We can relate error estimates for h and $\frac{h}{2}$:

$$E\left(f;\frac{h}{2}\right) \approx \frac{1}{2^q}E(f;h).$$

So, if $E(f;h) \approx Kh^q$, then
$$E\left(f;\frac{h}{2}\right) \approx \frac{Kh^q}{2^q}.$$

- Manipulating Error:

$$\begin{aligned}
E(f;h) &= I(f) - Q(f;h) \\
&= \underbrace{I(f) - Q\left(f;\frac{h}{2}\right)}_{E\left(f;\frac{h}{2}\right)} + Q\left(f;\frac{h}{2}\right) - Q(f;h) \\
&\approx \frac{1}{2^q}E(f;h) + \left(Q\left(f;\frac{h}{2}\right) - Q(f;h)\right). \\
E(f;h) &\approx \underbrace{\left(\frac{2^q}{2^q-1}\right)\left(Q\left(f;\frac{h}{2}\right) - Q(f;h)\right)}_{\substack{\text{a posteriori error estimate} \\ \text{(computable)}}}
\end{aligned}$$

- Implementation: Recursive Process. For each subinterval:
  1. check: $\left|Q\left(f;\frac{h}{2}\right) - Q(f;h)\right| <$ tolerance.
  2. If true: we are good;
  3. If false: we need to refine abscissae. Cut the subinterval in half and repeat.
  4. Stop when all subintervals satisfy the tolerance condition.

- Good implementation practice:
  1. Reuse computation
  2. Parallelism

8 Numerical ODEs

8.1 Differential Equations

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y), \qquad a \leq t \leq b. \tag{ODE}$$

- (ODE) is a non-autonomous equation since f depends on t.
- If $f(t, y) = f(y)$ is not dependent on t, we call it *autonomous*.

Example 8.1.1 Solving ODE Analytically

$$y' = -y + t, \qquad t \geq 0.$$

Solution 1.
A solution:

$$y(t) = t - 1 + \alpha e^{-t}.$$

This is a family of solutions. It is not unique as α can be anything. To verify this is the solution, we compute

$$y' = 1 - \alpha e^{-t} = -y + t.$$

To make the solution unique, we need an initial condition $y(0) = C$. □

Theorem 8.1.2 General Procedure to Solve ODEs

$$y(t) = C + \int_a^t f(s, y(s))\,\mathrm{d}s,$$

where C is a constant, and $\int_a^t f(s, y(s))\,\mathrm{d}s$ is the *numerical integrator*. t is a moving bound.

- Initial value problem (IVP):

$$y(a) \text{ is given} \implies C = y(a).$$

- Terminal value problem (TVP):

$$y(b) \text{ is given}$$

  This can be transformed into IVP using mapping: $\tau = b - t$ where $0 \leq \tau \leq a$. So,

$$y(\tau) = C - \int_0^\tau f(s, y(s))\,\mathrm{d}s.$$

- Boundary value problem (BVP): Given information about y at multiple time points.

Example 8.1.3 Go-To Example

$$y' = \lambda y, \quad y(0) = 1, \quad t \geq 0.$$

Solution: $y = e^{\lambda t}$.

8.2 Euler's Method

8.2.1 Approximate y_i, then update. Suppose we have an approximation $\underbrace{y(t_i)}_{\text{exact}} \approx \underbrace{y_i}_{\text{approx.}}$. What is $y(t_{i+1})$?

Assume $t_{i+1} = t_i + h$. Then, by Taylor's approximation,

$$y(t_{i+1}) = y(t_{i+h}) = y(t_i) + h \underbrace{y'(t_i)}_{=f(t_i, y(t_i))} + \frac{h^2}{2} y''(\xi_i).$$

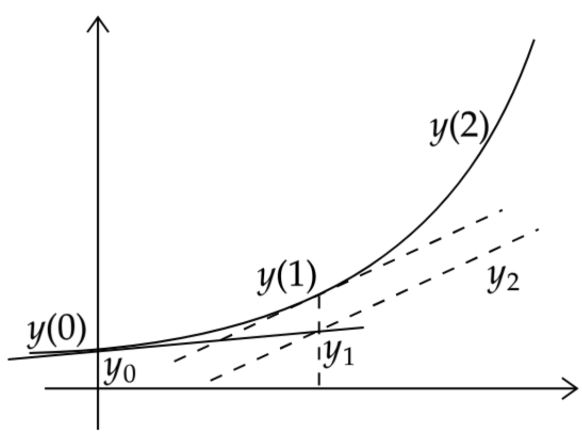
$$y_{i+1} = y_i + hf(t_i, y_i) \qquad \text{(Forward Euler)}$$

8.2.2 Approximate y_i', then update.

$$f(t_i, y_i) \approx \frac{y_{i+1} - y_i}{h} \qquad \text{[derivative approximation]}$$

Example 8.2.3 Test Problem

$$y' = \lambda y; \qquad y_{\text{ex}}(t) = y(0)e^{\lambda t}, \quad y(0) = 1, \ t > 0.$$



8.2.4 Explicit vs. Implicit Methods.

- Forward Euler: forward difference

$$f(t_i, y(t_i)) = y'(t_i) \approx \frac{y(t_{i+1}) - y(t_i)}{h}$$
$$y_{i+1} = y_i + hf(t_i, y_i) \tag{FE}$$

Explicit method: we can evaluate/compute. Only using information we have pre-computed.

(+) Faster to integrate

(+) Easy to implement

- Backward Euler: backward difference

$$f(t_{i+1}, y(t_{i+1})) = y'(t_{i+1}) \approx \frac{y(t_{i+1}) - y(t_i)}{h}$$
$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}) \tag{BE}$$

Implicit method: we cannot evaluate/compute. We are trying to solve for y_{i+1}. (we can use fixed point iteration or other root finding methods).

(+) Other numerical benefits.

Example 8.2.5 Test Problem

$$y' = \lambda y; \qquad y(0) = 1, \qquad t > 0.$$

- FE: $y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda)y_i$
- BE: $y_{i+1} = y_i + h\lambda y_{i+1} \implies y_{i+1} = \dfrac{1}{(1 - h\lambda)} y_i$.

8.3 Numerical Considerations in Euler's Method

Definition 8.3.1 (Local Truncation Error). The amount by which the exact solution fails to satisfy the difference equation at integration step i.

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - \underbrace{f'(t_i, y(t_i))}_{y'(t_i)}.$$

Remark. For FE: $d_i \sim \mathcal{O}(h)$. That is, if we cut step size by half, the local truncation error decreases by half.

Definition 8.3.2 (Order of Accuracy). The smallest positive integer q $s.t.$

$$\max_i |d_i| = \mathcal{O}(h^q).$$

Definition 8.3.3 (Global Error).

$$e_i = y(t_i) - y_i.$$

Remark. Generally, order of accuracy is the same as local truncation error (when we have nice functions). For example, for FE, $\max_i |e_i| \sim \mathcal{O}(h)$.

Definition 8.3.4 (Convergence). A numerical ODE integrator is said to *converge* if the maximum global error $\to 0$ when $h \to 0$.

Theorem 8.3.5 FE Convergence
Suppose:

- $f(t, y)$ have bounded partial derivatives in $\mathcal{D} = \{a \le t \le b, \quad |y| < \infty\}$.

  This implies Lipschitz continuity in y:

  $$|f(t, y) - f(t, \widehat{y})| \le L|y - \widehat{y}| \quad \forall\, (t, y), (t, \widehat{y}) \in \mathcal{D}.$$

- $y(t)$ has bounded second derivative:

  $$\|y''\|_\infty \le \text{constant}.$$

Then, FE converges and global error decreases linearly in h. i.e.,

$$\max_{i=0,\ldots,N} |e_i| = \max_{i=0,\ldots,N} |y(t_i) - y_i| \le Bh,$$

where $y(t_i)$ is the true solution, y_i is the approximation by FE ($y_i = y_{i-1} + hf(t_{i-1}, y_{i-1})$), and $B = \dfrac{e^{(b-a)L} - 1}{L} \cdot \dfrac{\|y''\|_\infty}{2}$ is a constant.

Proof 1.

$$
\begin{aligned}
e_i &= y(t_i) - y_i \\
d_i &= \frac{y(t_{i+1}) - y(t_i)}{h} - \overbrace{f(t_i, y(t_i))}^{y'(t_i)} = \frac{h}{2} y''(\xi_i) \qquad ① && \text{[Local truncation error (LTE)]} \\
d(h) &= \max_{i=0,\ldots,N} |d_i| \\
0 &= \frac{y_{i+1} - y_i}{h} - f(t_i, y_i) \qquad ② && \text{[from FE: } y_{i+1} = y_i + hf(t_i, y_i)\text{]} \\
① - ② : d_i &= \frac{y(t_{i+1}) - y(t_i)}{h} - f(t_i, y(t_i)) - \frac{y_{i+1} - y_i}{h} + f(t_i, y_i) \\
&= \frac{e_{i+1} - e_i}{h} - \big(f(t_i, y(t_i)) - f(t_i, y_i)\big)
\end{aligned}
$$

So,

$$
\begin{aligned}
e_{i+1} &= e_i + h\big(f(t_i, y(t_i)) - f(t_i, y_i)\big) + hd_i \\
|e_{i+1}| &= \big|e_i + h\big(f(t_i, y(t_i)) - f(t_i, y_i)\big) + hd_i\big| \\
&\leq |e_i| + h|f(t_i, y(t_i)) - f(t_i, y_i)| + h|d_i| && \text{[Triangle inequality]} \\
&\leq |e_i| + hL \underbrace{|y(t_i) - y_i|}_{|e_i|} + h|d_i| && \text{[Lipschitz]} \\
&= |e_i| + hL|e_i| + h|d_i| \\
&= (1 + hL)|e_i| + h|d_i| \\
&\leq (1 + hL)|e_i| + hd(h). && \left[d(h) = \max_{i=0,\ldots,N} |d_i|\right]
\end{aligned}
$$

If we iterate:

$$
\begin{aligned}
|e_{i+1}| &\leq (1 + hL)|e_i| + hd(h) \\
&\leq (1 + hL)\big[(1 + hL)|e_{i-1}| + hd(h)\big] + hd(h) \\
&= (1 + hL)^2|e_{i-1}| + hd(h)\big[1 + (1 + hd(h))\big] \\
&\vdots \\
&\leq \underbrace{(1 + hL)^{i+1}|e_0|} + hd(h) \cdot \sum_{k=0}^{i} (1 + hL)^k && \text{[with IVP: } e_0 = y(t_0) - y_0 = 0\text{]} \\
&= hd(h) \cdot \sum_{k=0}^{i} (1 + hL)^k \\
&= hd(h) \cdot \left(\frac{1 - (1 + hL)^i}{-hL}\right) = \frac{d(h)}{L}\big[(1 + hL)^i - 1\big]. && \left[\text{finite geometric sum: } \frac{1 - r^n}{(1 - r)}\right]
\end{aligned}
$$

Lemma 8.6 : For any real x:

$$1 + x \leq e^x$$

and if $x \geq -1$, then

$$0 \leq (1+x)^m \leq e^{mx}.$$

Proof. $e^x = 1 + x + \dfrac{x^2}{2} e^{\xi} > 1 + x.$ □

So, by this Lemma,

$$(1 + hL)^i \leq e^{ihL} \leq e^{NhL} = e^{(b-a)L}.$$

Further,

$$\begin{aligned} d(h) = \max_{i=0,\dots,N} |d_i| &= \max_{i=0,\dots,N} \left| \frac{h}{2} y''(\xi_i) \right| \\ &\leq \frac{h}{2} \|y''\|_\infty. \end{aligned}$$

Then,

$$\begin{aligned} |e_{i+1}| &\leq \frac{h}{2} \|y''\|_\infty \cdot \left[\frac{e^{(b-a)L} - 1}{L} \right] \\ &= \left[\frac{e^{(b-a)L} - 1}{L} \right] \cdot \frac{\|y''\|_\infty}{2} \cdot h \\ &\sim \mathcal{O}(h). \end{aligned}$$

■

8.4 Runge-Kutta Methods

Motivation: Higher order explicit method.

8.4.1 Implicit Trapezoidal Method.

$$y(t_{i+1}) = y(t_i) + \underbrace{\int_{t_i}^{t_{i+1}} f(s, y(s))\, \mathrm{d}s}_{\text{quadrature rules}} \qquad \text{(True solution)}$$

- Use trapezoidal rule for integrals:

$$\begin{aligned} \int_{t_i}^{t_{i+1}} f(s, y(s))\, \mathrm{d}s &= \frac{h}{2} \big(f(t_i, y_i) + f(t_{i+1}, y_{i+1}) \big) \\ y_{i+1} &= y_i + \frac{h}{2} \big(f(t_i, y_i) + f(t_{i+1}, y_{i+1}) \big). \end{aligned}$$

- **Claim 8.2** The LTE

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{1}{2}\big(\underbrace{f(t_i, y(t_i))}_{y'(t_i)} + \underbrace{f(t_{i+1}, y(t_{i+1}))}_{y'(t_{i+1})}\big)$$

is of order h^2.

Proof 1.

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{3!}y'''(\xi_i) \\ y'(t_{i+1}) &= y'(t_i) + hy''(t_i) + \frac{h^2}{2}y'''(\eta_i) && \text{(Taylor expansion on derivative)} \end{aligned}$$

Then,

$$\begin{aligned} d_i &= \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{1}{2}\big(f(t_i, y(t_i)) + f(t_{i+1}, y(t_{i+1}))\big) \\ &= \cancel{y'(t_i)} + \cancel{\frac{h}{2}y''(t_i)} + \frac{h^2}{3!}y'''(\xi_i) - \cancel{\frac{1}{2}y'(t_i)} - \cancel{\frac{1}{2}y'(t_i)} - \cancel{\frac{h}{2}y''(t_i)} - \frac{h^2}{4}y'''(\eta_i) \\ &= \frac{h^2}{3!}y'''(\xi_i) - \frac{h^2}{4}y'''(\eta_i) \\ &\sim \mathcal{O}\big(h^2\big). \end{aligned}$$

■

8.4.3 Explicit Trapezoidal Methods.

$$\begin{cases} \widehat{y}_{i+1} = y_i + hf(t_i, y_i) \\ y_{i+1} = y_i + \dfrac{h}{2}\big(f(t_i, y_i) + f(t_{i+1}, \widehat{y}_{i+1})\big) \end{cases}$$

Order: $\mathcal{O}(h^2)$.

8.4.4 Midpoint Methods.

- Implicit Midpoint:

$$\int_{t_i}^{t_{i+1}} f(s, y(s))\,\mathrm{d}s = hf(t_{i+1/2}, y_{i+1/2}),$$

where $t_{i+1/2} = \dfrac{t_i + t_{i+1}}{2}$ and $y_{i+1/2} = \dfrac{y_i + y_{i+1}}{2}$. So,

$$\begin{aligned} y_{i+1} &= y_i + hf\left(\frac{t_i + t_{i+1}}{2}, \frac{y_i + y_{i+1}}{2}\right) \\ &= y_i + hf(t_{i+1/2}, y_{i+1/2}). \end{aligned}$$

- Explicit Midpoint:

$$\begin{cases} \widehat{y}_{i+1/2} = y_i + \frac{h}{2} f(t_i, y_i) \\ y_{i+1} = y_i + h f(t_{i+1/2}, \widehat{y}_{i+1/2}) \end{cases}$$

  Explicit midpoint and explicit trapezoidal methods are 2 *stage* methods.

- Order: $\mathcal{O}(h^2)$

8.4.5 Runge-Kutta (RK) 4 Method.

$$\begin{aligned} Y_1 &= y_i & &\approx y(t_i) \\ Y_2 &= y_i + \frac{h}{2} f(t_i, Y_1) & &\approx y(t_{i+1/2}) \\ Y_3 &= y_i + \frac{h}{2} f(t_{i+1/2}, Y_2) & &\approx y(t_{i+1/2}) \\ Y_4 &= y_i + h f(t_{i+1/2}, Y_3) & &\approx y(t_{i+1}) \end{aligned}$$

$$y_{i+1} = y_i + \frac{h}{6}\big(f(t_i, Y_1) + 2f(t_{i+1/2}, Y_2) + 2f(t_{i+1/2}, Y_3) + f(t_{i+1}, Y_4)\big).$$

Order: $\mathcal{O}(h^4)$.

8.5 Absolute Stability and Stiffness

Definition 8.5.1 (Test Equation).

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}, \quad y(0) = y_0.$$

Exact solution: $y(t) = y_0 e^{\lambda t}$. (Recall: $e^{(a+bi)t} = e^{at}\big(\cos(bt) + \mathrm{i}\sin(bt)\big)$)

Definition 8.5.2 (Absolute Stability). A numerical integrator has *absolute stability* if the solution does not diverge in magnitude as $t \to \infty$. i.e.,

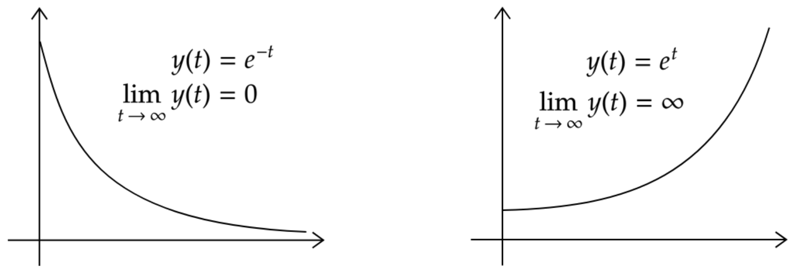
$$|y(t_{i+1})| \leq |y(t_i)| \qquad \text{eventually.}$$

Example 8.5.3

- In test problem:

$$|y(t)| = |y_0| e^{\mathrm{Re}(\lambda) t}.$$

  If $\mathrm{Re}(\lambda) \leq 0$, the solution is absolutely stable.



- FE stability:

$$y_{i+1} = y_i + hf(t_i, y_i).$$

For the test equation, we have

$$y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda)y_i.$$

To make it absolutely stable: $|y_{i+1}| \leq |y_i|$. This happens when

$$|1 + h\lambda| \leq 1.$$

This is the condition for absolute stability for FE.

1. $\lambda > 0$: no absolute stability at all.
2. $\lambda < 0$: need to choose h carefully to have absolute stability.

Definition 8.5.4 (Region of Stability). The set of complex numbers for which numerical solution is absolutely stable ($z = h\lambda \in \mathbb{C}$).

Example 8.5.5

- FE: $R = \{z \in \mathbb{C} : |1 + z| < 1\}$.
- BE:

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1})$$
$$y_{i+1} = \frac{1}{1 - h\lambda} y_i$$

Stability requires: $\left|\frac{1}{1-h\lambda}\right| \leq 1 \implies |1-h\lambda| \geq 1$.

Denote $z = h\lambda \in \mathbb{C}$. Then, the region of stability: $R = \{z \in \mathbb{C} : |1-z| \geq 1\}$.

- Some other explicit method (suspicious RK2 method):

$$\begin{aligned} \widehat{y}_{i+1} &= (1+h\lambda)y_i \\ y_{i+1} &= y_i + hf(t_{i+1}, \widehat{y}_{i+1}) \\ &= y_i + h\lambda(1+h\lambda)y_i \\ &= \big(1 + h\lambda + (h\lambda)^2\big)y_i \end{aligned}$$

Take $z = h\lambda \in \mathbb{C}$. Then, the region of stability is

$$R = \left\{z \in \mathbb{C} : \left|1 + z + z^2\right| \leq 1\right\}.$$

Definition 8.5.6 (A-Stable Method). If the region of stability contains the entire left-half plane, the method is called *A-stable*.

Example 8.5.7

- BE is A-stable.
- In general, implicit methods tend to have A-stable property, but they are hard to implement.

Example 8.5.8

Consider $y' = f(y)$, autonomous.

Suppose $y(t)$ and $\widehat{y}(t)$ are two solutions. If $y(t)$ and $\widehat{y}(t)$ are absolutely stable, then

$$\lim_{t\to\infty} \underbrace{y(t) - \widehat{y}(t)}_{w(t)} = 0.$$

Form a new ODE:

$$\begin{aligned} w(t) &= y(t) - \widehat{y}(t) \\ w'(t) &= y'(t) - \widehat{y}'(t) \\ &= f(y) - f(\widehat{y}). \end{aligned}$$

Using Taylor's expansion of $f(y)$ around $f(\widehat{y})$:

$$f(y) = f(\widehat{y}) + \frac{\partial f}{\partial y} w(t) + \text{higher order terms}$$

So,

$$w'(t) = \underbrace{\frac{\partial f}{\partial y}}_{=\lambda(t)} w(t) + \text{higher order terms}.$$

That is,

$$w'(t) = \lambda(t) w(t).$$

Punchline: the test equation can be applied to a more general setting.

Definition 8.5.9 (Stiffness). An IVP is *stiff* if the step size needed to maintain absolute stability of FE is much smaller than the step size needed to represent the solution accurately.

Example 8.5.10

$$y' = -1000(y - \cos(t)) - \sin(t), \qquad y(0) = 1.$$

- Exact solution: $y(t) = \cos(t)$.
- The solution looks good for $h = 0.1$ i.e., by plotting $y(t_i)$.
- However, for stability of FE, we look at $y' = -1000y$, we require $h = \frac{1}{500}$.
- So, this is a stiff problem.

Remark 1. (Connection Between Optimization and ODE).

$$x_{i+1} = x_i - \alpha \boldsymbol{\nabla} \varphi(x_i) \qquad \text{(Gradient Descent)}$$
$$x'(t) = -\boldsymbol{\nabla} \varphi(x_i) \qquad \text{(Gradient Flow)}$$

So, GD is a FE discretization to gradient flow. One can even try other methods to solve the gradient flow problem.