MATH 437 Notes

Jordan Hoffart

February 25, 2026

Contents

1 Lecture 1

1.1 Bisection method

Let $f(x)$ be a continuous function on the interval $[a, b]$.

Proposition 1.1 (Existence of roots)**.** *If $f(a)f(b) < 0$, then there is a point $p \in (a, b)$ such that $f(p) = 0$.*

Proof. If $f(a)f(b) < 0$, then $f(a)$ and $f(b)$ have opposite signs. That is, $f(a) > 0$ and $f(b) < 0$, or $f(a) < 0$ and $f(b) > 0$. By the Intermediate Value Theorem, f attains all possible values between $f(a)$ and $f(b)$. In particular, there is a point $p \in (a, b)$ where $f(p) = 0$. □

The bisection method is an algorithm to find the point p. The algorithm is as follows for the case that $f(a) < 0 < f(b)$.

Algorithm 1 Bisection method

```
1: $x_{\text{left}} := a$, $x_{\text{right}} := b$, $n := 0$, $x := (x_{\text{left}} + x_{\text{right}})/2$
2: while $|x_{\text{left}} - x_{\text{right}}| > \text{tol}$ and $n \leq n_{\text{max}}$ do
3:     if $f(x) = 0$ then
4:         return $x$
5:     else if $f(x) < 0$ then
6:         $x_{\text{left}} \leftarrow x$
7:     else
8:         $x_{\text{right}} \leftarrow x$
9:     end if
10:    $n \leftarrow n + 1$
11:    $x \leftarrow (x_{\text{left}} + x_{\text{right}})/2$
12: end while
13: return $x$
```

1.2 Fixed point methods

Given a continuous function $g(x)$, suppose we want to solve the equation $x = g(x)$. One possible iterative method is defined by

$$x_{n+1} = g(x_n), \tag{1}$$

where we provide a starting value x_0. Whether or not this converges to a solution as $n \to \infty$ depends on the properties of g and the starting value x_0.

Theorem 1.2 (Existence of fixed points)**.** *If $g(x) \in [a, b]$ for all $x \in [a, b]$, then g has a fixed point in $[a, b]$.*

Proof. Let $h(x) = g(x) - x$. Then $h(a) \leq 0$, $h(b) \geq 0$ and h is continuous. If $h(a) = 0$, then a is a fixed point of g. If $h(b) = 0$, then b is a fixed point of g. If $h(a)$ and $h(b)$ are both nonzero, then $h(a) < 0 < h(b)$. By the previous proposition, there exists $x_0 \in (a, b)$ such that $h(x_0) = 0$, i.e. $g(x_0) = x_0$. □

Theorem 1.3 (Convergence of fixed-point methods)**.** *Suppose g is differentiable, and there exists k such that $|g'(x)| \leq k < 1$ for all $x \in [a, b]$. Then, g has a unique fixed point, and the iterative method* (1) *converges to this point for any initial value $x_0 \in [a, b]$.*

Proof. By the Mean Value Theorem, for distinct $x, y \in [a, b]$, there exists $z \in [a, b]$ such that

$$g(x) - g(y) = g'(z)(x - y).$$

Therefore, since $|g'(z)| \leq k < 1$, we have that

$$|g(x) - g(y)| \leq k|x - y| < |x - y|$$

for all distinct $x, y \in [a, b]$.

Now, let $x_0 \in [a, b]$, and set $x_{n+1} = g(x_n)$ for all $n \geq 0$. From above, for all $n \geq 0$,

$$|x_{n+2} - x_{n+1}| = |g(x_{n+1}) - g(x_n)| \leq k|x_{n+1} - x_n|.$$

By repeating this, we have

$$|x_{n+2} - x_{n+1}| \leq k^{n+1}|x_1 - x_0|$$

for all n. Therefore, for any $m > n \geq 0$, by writing $m = n + (m - n)$,

$$\begin{aligned} |x_m - x_n| &\leq |x_{n+(m-n)} - x_{n+(m-n-1)}| + |x_{n+(m-n-1)} - x_{n+(m-n-2)}| + \cdots + |x_{n+1} - x_n| \\ &\leq (k^{m-n-1} + k^{m-n-2} + \cdots + k^n)|x_1 - x_0|. \end{aligned}$$

Since $k < 1$, the terms in the last inequality are the Cauchy tail of the convergent geometric series $\sum_i k^i$. Therefore, $|x_m - x_n| \to 0$ as $m, n \to \infty$, so the sequence $(x_n)_n$ is a Cauchy sequence of real numbers. Therefore, the sequence must converge to some number p.

Since $g(x_n) = x_{n+1}$ and g is continuous, taking limits of this equation yields $g(p) = p$, so p is a fixed point of g. If q is another fixed point of g, then, from above,

$$|p - q| = |g(p) - g(q)| < |p - q|,$$

which is a contradiction, so p is the only fixed point of g. □

Proposition 1.4 (Non-convergence of fixed point methods)**.** *Let g be continuously differentiable with a fixed point $g(p) = p$. If $|g'(p)| > 1$ and $x_0 \neq p$, then the fixed point iteration $x_{n+1} = g(x_n)$ will not converge to p.*

Proof. Suppose for the sake of contradiction that $x_n \to p$. Since g' is continuous, there is $\delta > 0$ such that $|g'(x)| > 1$ when $x \in (p - \delta, p + \delta)$. By the Mean Value Theorem,

$$x_{n+1} - p = g(x_n) - g(p) = g'(\xi_n)(x_n - p)$$

for some ξ_n between x_n and p. Since $x_n \to p$, the Squeeze Theorem implies that $\xi_n \to p$ as well. Therefore, there is $N > 0$ such that $\xi_n \in (p - \delta, p + \delta)$ for $n > N$, which implies that $|g'(\xi_n)| > 1$ when $n > N$. Therefore,

$$|x_{n+1} - p| = |g'(\xi_n)||x_n - p| > |x_n - p|$$

when $n > N$. In particular,

$$|x_{n+1} - p| = |g'(\xi_n)||x_n - p| > |x_{N+1} - p| =: \varepsilon_0$$

for all $n > N$. Now, if $\varepsilon_0 = 0$, then $x_{N+1} = p$, so $x_{n+1} = p$ for all $n > N$, which contradicts the inequality above. On the other hand, if $\varepsilon_0 > 0$, then, since $x_n \to p$, there is $M > N$ such that, when

$n > M$, $|x_{n+1} - p| < \varepsilon_0/2$. This also contradicts the inequality above. Therefore, in all cases, we reach a contradiction, so $x_n \not\to p$. $\square$

2 Lecture 2

2.1 Newton's method

Newton's method is a fixed-point method to find the roots of a differentiable function $f(x)$. It is defined by the following algorithm:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2}$$

If we set $g(x) = f(x)/f'(x)$, then the above equation is of the form (1), so that Newton's method is indeed a fixed-point method.

Lemma 2.1. *Suppose f is twice differentiable and $f'(p) \neq 0$. Let $g(x) = x - f(x)/f'(x)$.*

1. *p is a fixed point of g iff $f(p) = 0$.*
2. *If $f(p) = 0$, then $g'(p) = 0$.*

Proof.

1. If p is a fixed point of g, then $p = g(p) = p - f(p)/f'(p)$, so $f(p) = 0$. Conversely, if $f(p) = 0$, then $g(p) = p - f(p)/f'(p) = p$.
2. $$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$
   Since $f(p) = 0$, $g'(p) = 0$.

□

Theorem 2.2 (Convergence of Newton's method)**.** *Suppose f is twice differentiable, has a root at p and $f'(p) \neq 0$. For any initial value x_0 sufficiently close to p, Newton's method converges to p.*

Proof. We let $g(x) = f(x)/f'(x)$. Then, g is continuously differentiable, and, by the previous lemma, p is a fixed point of g and $g'(p) = 0$. Therefore, there exists $\delta > 0$ such that, whenever $|x - p| \leq \delta$, $|g'(x)| \leq 1/2 < 1$. By using Theorem 1.3 with $k = 1/2$, $a = p - \delta$, $b = p + \delta$, we conclude that whenever $x_0 \in [a, b]$, Newton's method converges to p. □

2.2 Quadratic convergence of Newton's method

Definition 2.3 (Order of convergence)**.** Suppose that a sequence $x_n \to p$ as $n \to \infty$. We say that the sequence converges with order $r > 0$ if there is a constant $0 \leq \lambda < \infty$ such that

$$|x_{n+1} - p| \leq \lambda |x_n - p|^r \tag{3}$$

for all n sufficiently large. For $r = 1$, we say the sequence converges linearly, and for $r = 2$, we say the sequence converges quadratically.

Theorem 2.4 (Quadratic convergence of Newton's method)**.** *Let f be a 3-times continuously differentiable function with a root at p and $f'(p) \neq 0$. Suppose that an initial value x_0 is chosen sufficiently close to p so that Newton's method converges to p. Then, the method converges quadratically.*

Proof. We let $g(x) = f(x)/f'(x)$. Then, g is a twice continuously differentiable function. Using Taylor's Theorem around p, for all n, there exists ξ_n between x_n and p such that

$$x_{n+1} = g(x_n) = g(p) + g'(p)(x_n - p) + \frac{g''(\xi_n)}{2}(x_n - p)^2.$$

From Lemma 2.1, $g(p) = p$ and $g'(p) = 0$, so

$$|x_{n+1} - p| = \frac{|g''(\xi_n)|}{2}|x_n - p|^2.$$

There exists $N > 0$ such that $|x_n - p| \leq 1$ for all $n \geq N$. Thus, for all $n \geq N$, ξ_n lies in the interval $[p-1, p+1]$. Since g'' is continuous on the closed and bounded interval $[p-1, p+1]$, we may set

$$\lambda := \max_{\xi \in [p-1,p+1]} \frac{|g''(\xi)|}{2}.$$

Then, we conclude that

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

when $n \geq N$, so Newton's method converges quadratically. □

2.3 Secant method

In Newton's method, we may replace $f'(x)$ by a backward difference approximation

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

Doing so gives us the secant method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(x_{n-1})}(x_n - x_{n-1}), \tag{4}$$

where now we must provide 2 initial conditions x_0, x_1.

3 Lecture 3

3.1 Polynomial interpolation

Definition 3.1 (Lagrange polynomials)**.** Given distinct points x_0, ..., x_n, the ith Lagrange polynomial constructed from these points is

$$L_i(x) := \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}. \tag{5}$$

Example 3.2 (Lagrange polynomial)**.** With $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, we have

$$\begin{cases} L_0(x) = \dfrac{(x-1)(x-2)}{(0-1)(0-2)}, \\ L_1(x) = \dfrac{(x-0)(x-2)}{(1-0)(1-2)}, \\ L_2(x) = \dfrac{(x-0)(x-1)}{(2-0)(2-1)}. \end{cases}$$

Theorem 3.3 (Properties of Lagrange polynomials)**.** *Given distinct points x_0, ..., x_n, let L_i be the ith Lagrange polynomial constructed from these points. Then L_i is the unique polynomial of degree n such that $L_i(x_j) = 0$ if $i \neq j$ and $L_i(x_i) = 1$. Furthermore, the set $\{L_i : i = 0, \ldots, n\}$ of Lagrange polynomials is a basis for the space P_n of polynomials of degree at most n.*

Proof. From (5), we see that L_i is a product of n monomial terms $(x - x_j)/(x_i - x_j)$, so it is a polynomial of degree n. Since each monomial term evaluates to 1 at x_i, $L_i(x_i) = 1$. Since at least one monomial term vanishes at x_j, $L_i(x_j) = 0$ when $j \neq i$. Now, suppose that p is another polynomial of degree n such that $p(x_i) = 1$ and $p(x_j) = 0$ when $j \neq i$. From the Fundamental Theorem of Algebra, since p has n roots at the x_j, p can be factored as

$$p(x) = a \prod_{j \neq i} (x - x_j)$$

for some $a \in \mathbb{R}$. Since $p(x_i) = 1$, inserting this into the equation above and solving for a gives

$$a = \prod_{j \neq i} \frac{1}{x_i - x_j}$$

Therefore,

$$p(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} = L_i(x),$$

so L_i is the unique polynomial of degree n with the above properties.

Now, we show that the set of Lagrange polynomials forms a basis of P_n. Suppose that

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

for some coefficients $c_i \in \mathbb{R}$. Evaluating at $x = x_i$ and using the properties of the Lagrange polynomials gives $c_i = 0$ for all i. Therefore, the L_i are linearly independent. Now, let $p \in P_n$ and set

$$q(x) = \sum_{j=0}^{n} p(x_j) L_j(x).$$

Then $q(x_i) = p(x_i)$ for all i by construction, and $q \in P_n$. Set $r = p - q$. Then $r \in P_n$ and r vanishes at $n+1$ distinct points $x_0, \ldots, x_n$. Therefore, by the Fundamental Theorem of Algebra, $r = 0$, so $p = q$. Thus, the L_i span all of P_n, so they are a basis of P_n. $\square$

Corollary 3.4 (Lagrange basis expansion)**.** *For the Lagrange basis $\{L_i : i = 0, \ldots, n\}$ of P_n constructed from distinct points x_0, ..., x_n, any polynomial $p \in P_n$ has the following basis expansion:*

$$p(x) = \sum_{j=0}^{n} p(x_j) L_j(x). \tag{6}$$

Theorem 3.5 (Polynomial interpolation)**.** *Given finitely many samples $f(x_0)$, $f(x_1)$, ..., $f(x_n)$ of a function $f(x)$ at distinct points x_0, ..., x_n, there is a unique polynomial $p_n(x)$ of degree n passing through the points:*

$$p_n(x_i) = f(x_i) \textit{ for all } i. \tag{7}$$

Proof. We use the Lagrange basis polynomials L_i defined by the points x_i and set

$$p_n(x) = \sum_{j=0}^{n} f(x_j) L_j(x). \tag{8}$$

From the results above, $p_n(x_i) = f(x_i)$ for all i, $p_n \in P_n$, and p_n is the only polynomial of degree n with these properties. $\square$

Definition 3.6 (Interpolating polynomial)**.** For a function f defined at distinct points x_0, ..., x_n, we call p_n defined by (8) the (Lagrange form of) the interpolating polynomial of f at the points x_i.

Lemma 3.7 (Generalization of Rolle's Theorem)**.** *Let f be an $n+1$ times differentiable function such that $f(x_0) = f(x_1) = \cdots = f(x_{n+1})$ at $n+2$ distinct points $x_0 < x_1 < \cdots < x_{n+1}$. Then, there is a point $\xi \in (x_0, x_{n+1})$ such that $f^{(n+1)}(\xi) = 0$.*

Proof. Applying Rolle's Theorem to each subinterval (x_i, x_{i+1}) gives $n+1$ distinct points $x_0^1 < x_1^1 < \cdots < x_n^1$ between x_0 and x_{n+1} such that all $f'(x_i^1) = 0$. We apply Rolle's theorem again now to the subintervals (x_i^1, x_{i+1}^1) to get n distinct points $x_0^2 < x_1^2 < \cdots < x_{n-1}^2$ between x_0 and x_{n+1} such that all $f''(x_i^2) = 0$. Proceeding inductively, we eventually conclude that there are two distinct points $x_0^n < x_1^n$ between x_0 and x_{n+1} such that $f^{(n)}(x_i^n) = 0$, and then we conclude that there is a single point $\xi \in (x_0, x_{n+1})$ such that $f^{(n+1)}(\xi) = 0$. $\square$

Lemma 3.8 (Polynomial derivatives)**.** *For any polynomial p of degree n, $p^{(n+1)}(x) = 0$. For any polynomial q of degree $n+1$ of the form*

$$q(x) = \prod_{j=0}^{n} (x - x_j),$$

$q^{(n+1)}(x) = (n+1)!$.

Proof. Expanding p in its monomial basis gives $p(x) = \sum_{i=0}^{n} c_i x^i$ for some coefficients c_i. Taking $n+1$ derivatives of the right-hand side makes all the monomial terms vanish, so we get the first result.

For the second result, expanding the product gives us a polynomial of the form

$$q(x) = x^{n+1} + r(x)$$

where r is a polynomial of degree n. Taking $n+1$ derivatives of the right hand side and using the previous result finishes the proof. □

Theorem 3.9 (Error of polynomial interpolation)**.** *Let f be $n+1$ times differentiable and let p be the degree n Lagrange interpolating polynomial of f at the distinct points $x_0 < x_1 < \cdots < x_n$. Then, for all x, there is a point $\xi_x \in (\min(x, x_0), \max(x, x_n))$ such that*

$$f(x) = p(x) + \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j). \tag{9}$$

Proof. Fix $x \notin \{x_0, \dots, x_n\}$ and let

$$g(t) = f(t) - p(t) - (f(x) - p(x)) \prod_{j=0}^{n} \frac{t - x_j}{x - x_j}.$$

We observe that $g(x) = 0$ and $g(x_j) = 0$ for $j = 0, \dots, n$. Therefore, by the previous lemmas, there is ξ_x such that

$$g^{(n+1)}(\xi_x) = f^{(n+1)}(\xi_x) - \frac{f(x) - p(x)}{\prod_{j=0}^{n} x - x_j}(n+1)! = 0.$$

Rearranging this equation finishes the proof. □

Example 3.10 (Error estimate)**.** We let $f(x) = \sin(x)$ and we estimate the error between f and the linear Lagrange interpolant at the points $x = 0$ and $x = \pi/2$:

$$p(x) = x.$$

Using the previous theorem,

$$\sin(x) = x - \frac{\sin(\xi_x)}{2}(x - 0)(x - \pi/2).$$

Therefore, for $0 \leq x \leq \pi/2$,

$$|\sin(x) - x| \leq \frac{1}{2} \max_{\xi \in [0, \pi/2]} |\sin(\xi)||x(x - \pi/2)| = \frac{1}{2} x(\pi/2 - x),$$

and thus

$$\max_{x \in [0, \pi/2]} |\sin(x) - x| \leq \max_{x \in \pi/2} \frac{1}{2} x(\pi/2 - x) = \frac{\pi^2}{32}.$$

□

4 Lecture 4

4.1 Divided differences

Definition 4.1 (Newton basis polynomials)**.** Given distinct points
$x_0, x_2, \dots, x_{n-1}$, the Newton basis polynomials are

$$\begin{cases} N_0(x) := 1, \\ N_j(x) := \prod_{i=0}^{j-1}(x - x_i), \quad j = 1, \dots, n. \end{cases} \tag{10}$$

Proposition 4.2. *The Newton basis polynomials are a basis of the space P_n of all polynomials of degree at most n.*

Proof. Suppose that

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

for some coefficients c_j. Evaluating at $x = x_0$ makes all the terms except $c_0 N_0(x) = c_0$ vanish, so $c_0 = 0$. Then, evaluating at $x = x_1$ makes all the remaining terms vanish except $c_1 n_1(x) = c_1(x_1 - x_2)$, so $c_1 = 0$. Repeating this argument for the remaining terms, we conclude that all $c_i = 0$. Therefore, the N_j are linearly independent. Since dim $P_n = n + 1$ and there are $n + 1$ N_j, we conclude that the set is a basis of P_n. □

Remark 4.3 (Computational cost)**.** Consider the following computations with respect to the Lagrange basis versus the Newton basis:

$$\begin{cases} p_1(x) := \sum_{j=0}^{n} c_j L_j(x), \\ p_2(x) := \sum_{j=0}^{n} c_j N_j(x). \end{cases}$$

To evaluate $p_1(x)$ using the Lagrange basis, one needs $(n + 1)(n + 2)$ multiplications. Indeed, each L_i requires $n + 1$ multiplications to evaluate at a point, assuming one precomputes and stores the denominator $(\prod_{j \neq i}(x_i - x_j))^{-1}$. Multiplying by the coefficient c_j costs another multiplication, so there are $n + 2$ multiplications required to evaluate each term in $p_1(x)$. Since there are $n + 1$ terms, we conclude that $p_1(x)$ requires $(n + 1)(n + 2)$ multiplications in total.

On the other hand, $p_2(x)$ only requires $(n + 1)(n + 2)/2$ multiplications in total. Indeed, the Newton basis requires j multiplications to evaluate $N_j(x)$. Multiplying each N_j by c_j adds 1 more multiplication. Thus, the total number of multiplications required is

$$\sum_{j=0}^{n}(j + 1) = (n + 1)(n + 2)/2.$$

Therefore, using the Newton basis to evaluate a polynomial of degree n requires half of the computational cost as with the Lagrange basis, which can provide significant increase in performance in practice. □

From the previous remark, it is computationally advantageous to use the Newton basis over the Lagrange basis. Given the values $f(x_0)$, ..., $f(x_n)$ of a function at distinct points x_0, ..., x_n, the interpolating polynomial in terms of the Lagrange basis is given by (8). How do we express this polynomial in terms of the Newton basis? That is, how do we compute the coefficients c_j such that

$$p(x) = \sum_{j=0}^{n} c_j N_j(x), \qquad p(x_i) = f(x_i) \text{ for all } i?$$

The method of divided differences answers this question. We first illustrate with an example.

Example 4.4 (Divided differences)**.** Let's construct the quadratic interpolant using points x_0, x_1, x_2 and values $f(x_0)$, $f(x_1)$, $f(x_2)$ in terms of the Newton basis. We have

$$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1), \qquad p(x_i) = f(x_i) \text{ for all } i.$$

At $x = x_0$, this implies that $f(x_0) = c_0$. At $x = x_1$, we have the equation

$$f(x_1) = f(x_0) + c_1(x_1 - x_0),$$

so

$$c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

For $i \neq j$, let

$$f[x_i, x_j] := \frac{f(x_j) - f(x_i)}{x_j - x_i}.$$

Then $c_1 = f[x_0, x_1]$. At $x = x_2$, we have the equation

$$f(x_2) = f(x_0) + f[x_0, x_1](x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1).$$

Now, we subtract $f(x_1)$ from both sides and divide by $x_2 - x_1$ to get

$$\begin{aligned} f[x_1, x_2] &= \frac{f(x_0) - f(x_1)}{x_2 - x_1} + f[x_0, x_1]\frac{x_2 - x_0}{x_2 - x_1} + c_2(x_2 - x_0) \\ &= -f[x_0, x_1]\frac{x_1 - x_0}{x_2 - x_1} + f[x_0, x_1]\frac{x_2 - x_0}{x_2 - x_1} + c_2(x_2 - x_0) \\ &= f[x_0, x_1] + c_2(x_2 - x_0). \end{aligned}$$

Therefore,

$$c_2 = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}.$$

Motivated by this, for distinct i, j, k, we set

$$f[x_i, x_j, x_k] := \frac{f[x_j, x_k] - f[x_i, x_j]}{x_k - x_i},$$

so that $c_2 = f[x_0, x_1, x_2]$. In conclusion, the coefficients with respect to the Newton basis are

$$p(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1).$$

□

This example generalizes to degree n as follows.

Theorem 4.5 (Newton coefficients)**.** *For distinct points $x_0, x_1, \dots, x_n$ and a function f, we recursively define*

$$f[x_i] := f(x_i), \qquad 0 \le i \le n, \tag{11}$$

and

$$f[x_i, \dots, x_{i+k}] := \frac{f[x_{i+1}, \dots, x_{i+k}] - f[x_i, \dots, x_{i+k-1}]}{x_{i+k} - x_i}, \qquad 0 \le i \le n-1,\ 1 \le k \le n-i. \tag{12}$$

Then, for the interpolating polynomial p to f at the points x_i, its coefficients in Newton form are

$$p(x) = \sum_{j=0}^{n} f[x_0, \dots, x_j] N_j(x). \tag{13}$$

Example 4.6 (Newton coefficients)**.** For $n = 3$, we have

$$\begin{aligned}
f[x_0, x_1, x_2, x_3] &= \frac{1}{x_3 - x_0}(f[x_1, x_2, x_3] - f[x_0, x_1, x_2]),\\
f[x_0, x_1, x_2] &= \frac{1}{x_2 - x_0}(f[x_1, x_2] - f[x_0, x_1]),\\
f[x_1, x_2, x_3] &= \frac{1}{x_3 - x_1}(f[x_2, x_3] - f[x_1, x_2]),\\
f[x_0, x_1] &= \frac{1}{x_1 - x_0}(f[x_1] - f[x_0]),\\
f[x_1, x_2] &= \frac{1}{x_2 - x_1}(f[x_2] - f[x_1]),\\
f[x_2, x_3] &= \frac{1}{x_3 - x_2}(f[x_3] - f[x_2]),
\end{aligned}$$

so

$$\begin{aligned}
p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)&\\
+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2).&
\end{aligned}$$

□

The following table let's us easily build up the divided difference coefficients starting from the function values $f(x_i)$. To be explicit, we give the table for $n = 3$.

	0	1	2	3
0	$f(x_0)$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
1	$f(x_1)$	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	
2	$f(x_2)$	$f[x_2, x_3]$		
3	$f(x_3)$			

The entries in the table are computed one column at a time from left to right. In column 1 onward, an entry in the (i, j) position of the table is computed from by using the entries in positions $(i, j - 1)$ and $(i + 1, j - 1)$.

4.2 Hermite interpolation

Suppose that we want to find a polynomial $p(x)$ of minimal degree that interpolates a function f and its derivative f' at a finite set of points x_i:

$$p(x_i) = f(x_i), \qquad p'(x_i) = f'(x_i) \text{ for all } i.$$

Such a polynomial is called a Hermite interpolating polynomial. It can be computed using the method of divided differences. We first explain with an example.

Example 4.7 (Hermite polynomial)**.** We desire a polynomial $p(x)$ that interpolates $f(x_0)$ and $f'(x_0)$. Since we have 2 constraints, we consider a polynomial of degree 1:

$$p(x) = c_0 + c_1 x.$$

We now compute the coefficients c_0, c_1 that satisfy the constraints. First, $p'(x_0) = c_1$, so we set $c_1 = f'(x_0)$. Then, $p(x_0) = c_0 + f'(x_0)x_0$, so we set

$$c_0 = f(x_0) - f'(x_0)x_0.$$

Thus, after substituting and rearranging,

$$p(x) = f(x_0) + f'(x_0)(x - x_0).$$

□

Notice that, in the previous example, the interpolating polynomial is in Newton form with coefficients $f(x_0)$ and $f'(x_0)$:

$$p(x) = f(x_0)N_0(x) + f'(x_0)N_1(x).$$

In fact, by modifying the divided difference table, we can use the divided difference algorithm to compute the coefficients of the Hermite polynomial as follows. For the example above, we set $z_0 = z_1 = x_0$. Then, we construct the divided difference table using the $f(z_i)$, values:

		0	1
0	x_0	$f(z_0)$	$f[z_0, z_1]$
1	x_0	$f(z_1)$	

However, since $z_0 = z_1 = x_0$, $f(z_0) = f(z_1)$, so the difference $f[z_0, z_1]$ is undefined. In this case, we use the derivative $f'(z_0)$:

$$f[z_i, z_j] := f'(z_i) \quad \text{if } z_i = z_j. \tag{14}$$

From another viewpoint, we recall that the divided difference is formally defined as

$$f[z_i, z_j] = \frac{f(z_j) - f(z_i)}{z_j - z_i},$$

so that in the limit $z_j \to z_i$, we recover the derivative of f at z_i:

$$\lim_{z_j \to z_i} f[z_i, z_j] = f'(z_i).$$

In the general case with $n+1$ points $x_0, \dots, x_n$ where we wish to interpolate the point values $f(x_i)$ and the derivatives $f'(x_i)$, the above procedure generalizes to the following table:

		0	1	$\cdots$	$2n+2$
0	x_0	$f(z_0)$	$f[z_0, z_1]$	$\cdots$	$f[z_0, \dots, z_{2n+1}]$
1	x_0	$f(z_1)$	$f[z_1, z_2]$	$\cdots$	
2	x_1	$f(z_2)$	$f[z_2, z_3]$	$\cdots$	
3	x_1	$f(z_3)$	$f[z_3, z_4]$	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	
$2n$	x_n	$f(z_{2n})$	$f[z_{2n}, z_{2n+1}]$		
$2n+1$	x_n	$f(z_{2n+1})$			

where we use (14) whenever $z_i = z_j$. The Hermite interpolating polynomial in Newton form is then

$$p(x) = \sum_{j=0}^{2n+1} f[z_0, \dots, z_j] N_j(x). \tag{15}$$

We remark that the Hermite interpolating polynomial belongs to P_{2n+1} to satisfy the $2n+2$ constraints.

We can also compute the Hermite interpolating polynomial using the Lagrange basis polynomials L_i. The idea is to write the polynomial as

$$p(x) = \sum_{j=0}^{n} f(x_j) H_j(x) + \sum_{j=0}^{n} f'(x_j) \widehat{H}_j(x), \tag{16}$$

where H_j and $\widehat{H}_j$ are polynomials in P_{2n+1} such that

$$\begin{aligned} H_j(x_i) &= \delta_{ij}, \quad H_j'(x_i) = 0, \\ \widehat{H}_j(x_i) &= 0, \qquad \widehat{H}_j'(x_i) = \delta_{ij}, \end{aligned} \tag{17}$$

and δ_{ij} is the Kronecker delta.

Lemma 4.8. *If a polynomial $p \in P_n$ satisfies $p(x_0) = 0$ and $p'(x_0) = 0$, then there is a polynomial $q \in P_{n-2}$ such that*

$$p(x) = (x - x_0)^2 q(x).$$

Proof. From the Fundamental Theorem of Algebra, $p(x) = (x - x_0) r(x)$ for some $r \in P_{n-1}$. Then, $p'(x) = r(x) + (x - x_0) r'(x)$, so $p'(x_0) = r(x_0) = 0$. Thus, by the Fundamental Theorem of Algebra again, there is $q \in P_{n-2}$ such that $r(x) = (x - x_0) q(x)$. Therefore, $p(x) = (x - x_0)^2 q(x)$ as desired. $\square$

Theorem 4.9 (Lagrange form of Hermite interpolating polynomial)**.** *The Lagrange form of the Hermite interpolating polynomial to the function f interpolating the values $f(x_0),\dots,f(x_n)$ and derivatives $f'(x_0),\dots,f'(x_n)$ is* (16) *with coefficients*

$$\begin{cases} H_j(x) = L_j(x)^2 (1 - 2L_j'(x_j)(x - x_j)), \\ \widehat{H}_j(x) = L_j(x)^2 (x - x_j), \end{cases} \tag{18}$$

where the L_j are the Lagrange basis polynomials constructed from the points $x_0,\dots,x_n$.

Proof. Following the idea in (17), we seek $H_j \in P_{2n+1}$ with the aforementioned properties. We show that necessarily the H_j must be of the form (18).

From the Fundamental Theorem of Algebra, H_j must be of the form

$$H_j(x) = q_j(x) \prod_{i \neq j} (x - x_i)$$

for some $q_j \in P_{n+1}$. By multiplying and dividing by $\prod_{i \neq j}(x_j - x_i)$ and redefining q_j, H_j is of the form

$$H_j(x) = q_j(x) L_j(x).$$

We observe that $L_j'(x_i) \neq 0$ for all $i \neq j$. Therefore,

$$H_j'(x_i) = q_j'(x_i)L_j(x_i) + q_j(x_i)L_j'(x_i) = q_j(x_i)L_j'(x_i) = 0$$

for all $i \neq j$, so $q_j(x_i) = 0$ for all $i \neq j$. By repeating the previous argument, we conclude that q_j must also be of the form

$$q_j(x) = L_j(x) r_j(x)$$

for some $r_j \in P_1$. Thus,

$$H_j(x) = L_j^2(x) r_j(x).$$

We desire also that $H_j(x_j) = 1$, so $r_j(x_j) = 1$. Thus, from the Fundamental Theorem of Algebra, there is a constant c_j such that

$$r_j(x) = 1 + c_j(x - x_j).$$

We also require that $H_j'(x_j) = 0$, so

$$H_j'(x_j) = 2L_j(x_j)L_j'(x_j)r_j(x_j) + L_j^2(x_j)r_j'(x_j) = 2L_j'(x_j) + c_j = 0.$$

Thus, $c_j = -2L_j'(x_j)$, hence

$$H_j(x) = L_j^2(x)(1 - 2L_j'(x_j)(x - x_j))$$

as desired.

Now, we do a similar procedure for $\widehat{H}_j$. Since $\widehat{H}_j(x_i) = 0$ for all i, we must have that

$$\widehat{H}_j(x) = L_j(x)(x - x_j)s_j(x)$$

for some $s_j \in P_n$. Since $\widehat{H}_j'(x_i) = 0$ for $i \neq j$, we have that

$$\begin{aligned} \widehat{H}_j'(x_i) &= L_j'(x_i)(x_i - x_j)s_j(x_i) + L_j(x_i)s_j(x_i) + L_j(x_i)(x_i - x_j)s_j'(x_i) \\ &= L_j'(x_i)(x_i - x_j)s_j(x_i) \\ &= 0. \end{aligned}$$

Thus, $s_j(x_i) = 0$ for all $i \neq j$. Hence, there is a constant a_j such that

$$\widehat{H}_j(x) = a_j L_j^2(x)(x - x_j).$$

Thus,

$$\widehat{H}_j'(x) = 2a_j L_j'(x)(x - x_j) + a_j L_j^2(x).$$

Since $\widehat{H}_j'(x_j) = 1$, this implies that $a_j = 1$, which completes the proof. $\square$

5 Lecture 5

5.1 Cubic splines

Definition 5.1 (Cubic spline)**.** Let $a = x_0 < x_1 < \cdots < x_n = b$. The cubic spline interpolating a function f at the points x_j is the piecewise cubic polynomial S with the following properties:

1. On each subinterval $[x_j, x_{j+1}]$, S is a cubic polynomial. We denote the restriction of S to this subinterval by S_j.

2. The pieces $\{S_j\}_{j=0}^{n-1}$ interpolate f at the nodes:

$$S_j(x_j) = f(x_j) \text{ and } S_j(x_{j+1}) = f(x_{j+1}) \text{ for all } j. \tag{19}$$

   This implies that S is continuous on the entire interval $[a, b]$, since $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$ for all j.

3. The first and second derivatives of S are continuous on $[a, b]$, i.e.

$$\begin{cases} S_j'(x_{j+1}) = S_{j+1}'(x_{j+1}), \\ S_j''(x_{j+1}) = S_{j+1}''(x_{j+1}) \end{cases} \tag{20}$$

   for all $j = 0, \ldots, n-2$.

4. S has natural boundary conditions:

$$S''(a) = S''(b) = 0. \tag{21}$$

Example 5.2 (Cubic spline)**.** Let's construct the cubic spline passing through the points (x_0, y_0), (x_1, y_1), and (x_2, y_2) where $x_0 < x_1 < x_2$. Since there are 3 points, $n = 2$, so there are two pieces S_0 and S_1. Since S_0'' and S_1'' are degree 1 polynomials, let's expand them in terms of the Lagrange basis on their respective subintervals:

$$\begin{cases} S_0''(x) = S_0''(x_0)\dfrac{x - x_1}{x_0 - x_1} + S_0''(x_1)\dfrac{x - x_0}{x_1 - x_0}, \\ S_1''(x) = S_1''(x_1)\dfrac{x - x_2}{x_1 - x_2} + S_1''(x_2)\dfrac{x - x_1}{x_2 - x_1}. \end{cases}$$

Now, let's integrate both functions on their respective subintervals starting from x_1:

$$\begin{cases} S_0'(x) = S_0'(x_1) + S_0''(x_0)\dfrac{(x - x_1)^2}{2(x_0 - x_1)} + S_0''(x_1)\dfrac{(x - x_0)^2 - (x_1 - x_0)^2}{2(x_1 - x_0)}, \\ S_1'(x) = S_1'(x_1) + S_1''(x_1)\dfrac{(x - x_2)^2 - (x_1 - x_2)^2}{2(x_1 - x_2)} + S_1''(x_2)\dfrac{(x - x_1)^2}{2(x_2 - x_1)}. \end{cases}$$

Then, we integrate again, also from x_1:

$$\begin{cases} S_0(x) = S_0(x_1) + S_0'(x_1)(x-x_1) + S_0''(x_0)\dfrac{(x-x_1)^3}{6(x_0-x_1)} \\ \qquad\qquad + S_0''(x_1)\dfrac{(x-x_0)^3-(x_1-x_0)^3-3(x_1-x_0)^2(x-x_1)}{6(x_1-x_0)}, \\ S_1(x) = S_1(x_1) + S_1'(x_1)(x-x_1) \\ \qquad\qquad + S_1''(x_1)\dfrac{(x-x_2)^3-(x_1-x_2)^3-3(x_1-x_2)^2(x-x_1)}{6(x_1-x_2)} \\ \qquad\qquad + S_1''(x_2)\dfrac{(x-x_1)^3}{6(x_2-x_1)}. \end{cases}$$

Now, we require $S_0(x_0) = y_0$, $S_0(x_1) = S_1(x_1) = y_1$, and $S_1(x_2) = y_2$. We also require $S_0'(x_1) = S_1'(x_1)$, $S_0''(x_1) = S_1''(x_1)$, $S_0''(x_0) = 0$, and $S_1''(x_2) = 0$. Inserting this above and evaluating the first equation at x_0 and the second at x_2 gives

$$\begin{cases} y_0 = y_1 + S_0'(x_1)(x_0-x_1) + S_0''(x_1)\dfrac{(x_1-x_0)^2}{2}, \\ y_2 = y_1 + S_0'(x_1)(x_2-x_1) + S_0''(x_1)\dfrac{(x_1-x_2)^2}{2}. \end{cases}$$

This is a square linear system in the unknowns $S_0'(x_1)$ and $S_0''(x_1)$. Subtracting each equation by y_1 and dividing by the respective coefficient in front of $S_0'(x_1)$ gives

$$\begin{cases} [y_0, y_1] = S_0'(x_1) + S_0''(x_1)\dfrac{(x_1-x_0)}{2}, \\ [y_1, y_2] = S_0'(x_1) + S_0''(x_1)\dfrac{(x_2-x_1)}{2}, \end{cases}$$

where $[y_i, y_j] := (y_i - y_j)/(x_i - x_j)$ is the divided difference of y_i and y_j. Subtracting the first equation from the second and dividing by $(x_2 - x_0)/2$ gives

$$S_0''(x_1) = 2[y_0, y_1, y_2] = S_1''(x_1),$$

where $[y_0, y_1, y_2] := ([y_1, y_2] - [y_0, y_1])/(x_2 - x_0)$ is the divided difference of the y_i's. If we now multiply the first equation by $x_2 - x_1$, the second by $x_1 - x_0$, and subtract the second from the first, we get

$$S_0'(x_1) = \frac{[y_0, y_1](x_2-x_1) - [y_1, y_2](x_1-x_0)}{x_2-x_0} = S_1'(x_1).$$

We have now found the unknowns $S_0'(x_1)$ and $S_0''(x_1)$ in terms of the data (x_i, y_i). We can use the Taylor expansions of the polynomials S_j from x_1 to give their formulas in terms of these coefficients:

$$\begin{cases} S_0(x) = y_1 + \dfrac{[y_0, y_1](x_2 - x_1) - [y_1, y_2](x_1 - x_0)}{x_2 - x_0}(x - x_1) \\ \qquad\qquad + [y_0, y_1, y_2](x - x_1)^2 + \dfrac{S_0^{(3)}(x_1)}{6}(x - x_1)^3, \\ S_1(x) = y_1 + \dfrac{[y_0, y_1](x_2 - x_1) - [y_1, y_2](x_1 - x_0)}{x_2 - x_0}(x - x_1) \\ \qquad\qquad + [y_0, y_1, y_2](x - x_1)^2 + \dfrac{S_1^{(3)}(x_1)}{6}(x - x_1)^3, \end{cases}$$

To compute $S_0^{(3)}(x_1)$ and $S_1^{(3)}(x_1)$, we return to the very first equations in the example and take derivatives, applying the boundary conditions and the previous results:

$$\begin{cases} S_0^{(3)}(x_1) = \dfrac{S_0''(x_1)}{x_1 - x_0} = \dfrac{2}{x_1 - x_0}[y_0, y_1, y_2], \\ S_1^{(3)}(x_1) = \dfrac{S_1''(x_1)}{x_1 - x_2} = \dfrac{2}{x_1 - x_2}[y_0, y_1, y_2]. \end{cases}$$

In conclusion, the general form for a piecewise cubic spline passing through 3 points is

$$\begin{cases} S_0(x) = y_1 + \dfrac{[y_0, y_1](x_2 - x_1) - [y_1, y_2](x_1 - x_0)}{x_2 - x_0}(x - x_1) \\ \qquad\qquad + [y_0, y_1, y_2](x - x_1)^2 \left(1 - \dfrac{1}{3}\dfrac{x_1 - x}{x_1 - x_0}\right), \\ S_1(x) = y_1 + \dfrac{[y_0, y_1](x_2 - x_1) - [y_1, y_2](x_1 - x_0)}{x_2 - x_0}(x - x_1) \\ \qquad\qquad + [y_0, y_1, y_2](x - x_1)^2 \left(1 - \dfrac{1}{3}\dfrac{x - x_1}{x_2 - x_1}\right), \end{cases}$$

where S_0 is defined on $[x_0, x_1]$ and $S_1$is defined on $[x_1, x_2]$. □

Remark 5.3. The procedure in the previous example does not easily generalize to more nodes. In this case, a more straightforward procedure involves simply expanding each piece S_j in some basis and setting up a large linear system imposed by the constraints. Motivated by the previous example where the final answer was expressed in terms of powers of $x - x_1$, we propose to use the following basis expansion for each S_j on $[x_j, x_{j+1}]$:

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3.$$

In fact, this is just a Taylor expansion of each S_j about x_j, so

$$\begin{cases} a_j := S_j(x_j), \\ b_j := S_j'(x_j), \\ c_j := \dfrac{S_j''(x_j)}{2}, \\ d_j := \dfrac{S_j^{(3)}(x_j)}{6}. \end{cases}$$

The compatibility conditions then become

$$\begin{cases} y_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \\ \quad = a_j + b_j(x_{j+1}-x_j) + c_j(x_{j+1}-x_j)^2 + d_j(x_{j+1}-x_j)^3, \\ b_{j+1} = S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \\ \quad = b_j + 2c_j(x_{j+1}-x_j) + 3d_j(x_{j+1}-x_j)^2, \\ 2c_{j+1} = S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) = 2c_j + 6d_j(x_{j+1}-x_j), \\ a_j = S_j(x_j) = y_j, \\ c_0 = S''_0(x_0) = 0, \\ S''_{n-1}(x_n) = 0 = 2c_{n-1} + 6d_{n-1}(x_n - x_{n-1}). \end{cases}$$

These equations lead to a large linear system that is no longer easily solvable by hand, but the system is simple to set up for a particular n and given values (x_j, y_j). □

5.2 Numerical differentiation

We recall the limit definition of a derivative:

$$f'(x_0) := \lim_{h\to 0} \frac{f(x_0+h) - f(x_0)}{h}. \tag{22}$$

We expect that, for h small enough, the divided difference on the right-hand side is a good approximation to $f'(x_0)$.

Theorem 5.4 (First-order numerical differentiation)**.** *If f is twice differentiable on an interval $(x_0-\delta, x_0+\delta)$ and f'' is bounded on that interval, then for all h small enough*

$$\left| f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} \right| \le \sup_{x\in(x_0-\delta,x_0+\delta)} \left| \frac{f''(\xi)}{2} \right| h. \tag{23}$$

Proof. We perform a Taylor expansion near x_0:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(\xi_h)$$

where ξ_h is between x_0 and x_0+h and h is small enough so $x_0 + h \in (x_0-\delta, x_0+\delta)$. Rearranging, taking absolute values, and taking a supremum over $\xi_h \in (x_0-\delta, x_0+\delta)$ finishes the proof. □

Thus, the numerical differentiation formula above is only first-order accurate. To achieve higher-order accurate formulas, we can use more points along with Taylor expansions or Lagrange expansions.

Example 5.5 (Lagrange expansion)**.** Consider 3 points x_0, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$. The Lagrange polynomial interpolating a smooth function f through these points is

$$p(x) = f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.$$

From Theorem 3.9, for all $x \in (x_0, x_2)$, there is a point $\xi_x \in (x_0, x_2)$ such that

$$f(x) = p(x) + \frac{f^{(3)}(\xi_x)}{6}(x - x_0)(x - x_1)(x - x_2).$$

Since f and p are smooth, we deduce that the mapping $x \mapsto f^{(3)}(\xi_x)$ is smooth on (x_0, x_1) and (x_1, x_2). Then, we may differentiate both sides to get

$$f'(x) = p'(x) + \frac{\mathrm{d}}{\mathrm{d}x}\frac{f^{(3)}(\xi_x)}{6}(x - x_0)(x - x_1)(x - x_2) \\ + \frac{f^{(3)}(\xi_x)}{6}\frac{\mathrm{d}}{\mathrm{d}x}\left\{(x - x_0)(x - x_1)(x - x_2)\right\}.$$

Evaluating at $x = x_0$ and substituting for h gives

$$f'(x_0) = f(x_0)\frac{-3h}{(-h)(-2h)} + f(x_1)\frac{-2h}{h(-h)} + f(x_2)\frac{-h}{(2h)h} + \frac{f^{(3)}(\xi_0)}{6}(-h)(-2h).$$

Rearranging, we arrive at a differentiation formula that is second-order accurate:

$$f'(x_0) = \frac{-3}{2h}f(x_0) + \frac{2}{h}f(x_0 + h) - \frac{1}{2h}f(x_0 + 2h) + \frac{f^{(3)}(\xi_0)}{3}h^2. \tag{24}$$

If we evaluate at x_1 instead, we get another formula:

$$f'(x_1) = \frac{f(x_1 + h) - f(x_1 - h)}{2h} - \frac{f^{(3)}(\xi_1)}{6}h^2. \tag{25}$$

□

We can also prove that the differentiation formulas are second-order accurate by using Taylor's theorem. We prove one of them and leave the other as an exercise to the reader.

Theorem 5.6 (Second-order differentiation formula)**.** *The differentiation formula* (25) *is second-order accurate for sufficiently smooth functions.*

Proof. We take $h > 0$ and expand in Taylor series about x_1:

$$\begin{cases} f(x_1 + h) = f(x_1) + hf'(x_1) + \dfrac{h^2}{2}f''(x_1) + \dfrac{f^{(3)}(\xi_{h,+})}{6}h^3, \\ f(x_1 - h) = f(x_1) - hf'(x_1) + \dfrac{h^2}{2}f''(x_1) - \dfrac{f^{(3)}(\xi_{h,-})}{6}h^3, \end{cases}$$

where $\xi_{h,+} \in (x_1, x_1 + h)$ and $\xi_{h,-} \in (x_1 - h, x_1)$. We subtract the two equations and rearrange:

$$\frac{f(x_1 + h) - f(x_1 - h)}{2h} - f'(x_1) = \frac{f^{(3)}(\xi_{h,+}) + f^{(3)}(\xi_{h,-})}{6}h^2.$$

Now, assuming that f is smooth enough where its 3rd derivative is bounded near x_1, by taking absolute values, we can bound the coefficient in front of the h^2 term on the right-hand side by a constant independent of h. This completes the proof. □

To summarize, using Lagrange polynomials is an effective way to discover differentiation formulas, and using Taylor's Theorem is an efficient way to prove that the methods are indeed accurate, provided that the functions involved are sufficiently smooth.

5.3 Richardson extrapolation

Sometimes, one can use a differentiation formula evaluated at multiple values of h to improve accuracy. This process is called Richardson extrapolation.

For example, consider the first-order formula to approximate $f'(x_0)$:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} + \mathcal{O}(h).$$

Denote the left-hand side by M and the differentiation formula on the right-hand side by $N(h)$.

Then, suppose that we can expand the error $M - N(h)$ as a power series:

$$M - N(h) = \sum_{j=1}^{\infty} K_j h^j = K_1 h + K_2 h^2 + \cdots .$$

Then,

$$2(M - N(h/2)) = K_1 h + 2K_2 (h/2)^2 + \cdots .$$

Subtracting these equations eliminates the $\mathcal{O}(h)$ term:

$$M - (2N(h/2) - N(h)) = \mathcal{O}(h^2).$$

We see that, by reusing the first-order accurate formula at h and $h/2$ and combining the results in a clever way, we formally achieve a second-order accurate formula:

$$\begin{aligned} f'(x_0) &= 2\frac{f(x_0+h/2) - f(x_0)}{h/2} - \frac{f(x_0+h) - f(x_0)}{h} \\ &= \frac{-3f(x_0) + 4f(x_0+h/2) - f(x_0+h)}{h}. \end{aligned} \tag{26}$$

Notice that, in this particular case, we re-derive (24) but with $h/2$ instead of h. However, by using more evaluations with different h's and some clever algebraic manipulations, even higher-order methods can be derived.

6 Lecture 6

6.1 Numerical integration

Example 6.1 (Trapezoid rule)**.** Consider a continuous function f defined on an interval $[x_0, x_1]$. Consider the Lagrange interpolating polynomial

$$p(x) = f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0}.$$

Suppose that f is twice continuously differentiable. We recall that, for $x \in (x_0, x_1)$, there is $\xi_x \in (x_0, x_1)$ such that

$$f(x) = p(x) + \frac{f''(\xi_x)}{2}(x - x_0)(x - x_1).$$

Since f and p are smooth, we conclude that the mapping $x \mapsto f''(\xi_x)$ is continuous on (x_0, x_1). Then, we integrate both sides:

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = \frac{f(x_0) + f(x_1)}{2}(x_1 - x_0) + \int_{x_0}^{x_1} \frac{f''(\xi_x)}{2}(x - x_0)(x - x_1)\,\mathrm{d}x.$$

The first term on the right is a formula for approximating the integral of f called the Trapezoid rule. Such a rule is called a quadrature rule. To bound the error of the quadrature rule, we subtract it from both sides of the previous equation and take absolute values:

$$\left|\int_{x_0}^{x_1} f(x)\,\mathrm{d}x - \frac{f(x_0) + f(x_1)}{2}(x_1 - x_0)\right| \le \frac{1}{2}\int_{x_0}^{x_1} |f''(\xi_x)|(x - x_0)(x_1 - x)\,\mathrm{d}x.$$

Since f'' is continuous on the closed and bounded interval $[x_0, x_1]$,

$$\begin{aligned}\left|\int_{x_0}^{x_1} f(x)\,\mathrm{d}x - \frac{f(x_0) + f(x_1)}{2}(x_1 - x_0)\right| &\\ \le \frac{1}{2}\max_{\xi\in[x_0,x_1]}|f''(\xi)|\int_{x_0}^{x_1}(x - x_0)(x_1 - x)\,\mathrm{d}x &\\ = \frac{1}{12}\max_{\xi\in[x_0,x_1]}|f''(\xi)|(x_1 - x_0)^3.&\end{aligned}$$

Thus, if we let $h := x_1 - x_0$, we see that the Trapezoid rule is formally third order accurate at approximating the integral of f. □

Similar to numerical differentiation, we used the Lagrange interpolating polynomial to derive a quadrature formula. Just like the previous section, we can also use Taylor's theorem to prove that the quadrature rule is third order accurate.

Theorem 6.2 (Trapezoid rule)**.** *The trapezoid rule*

$$\int_{x_0}^{x_0+h} f(x)\,\mathrm{d}x \approx \frac{h}{2}(f(x_0) + f(x_0 + h)) \tag{27}$$

provides a third order accurate approximation to the integral of a smooth function.

Proof. We take a Taylor expansion of f about x_0 to $x \in (x_0, x_0+h)$:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(\xi_x)}{2}(x-x_0)^2.$$

Then, we integrate both sides:

$$\int_{x_0}^{x_0+h} f(x)\,\mathrm{d}x = hf(x_0) + \frac{h^2}{2}f'(x_0) + \frac{1}{2}\int_{x_0}^{x_0+h} f''(\xi_x)(x-x_0)^2\,\mathrm{d}x.$$

Now, we replace $f'(x_0)$ by the following differentiation formula from the previous section:

$$f'(x_0) = \frac{f(x_0+h)-f(x_0)}{h} - \frac{h}{2}f''(\xi_0).$$

Inserting this above gives

$$\int_{x_0}^{x_0+h} f(x)\,\mathrm{d}x = \frac{h}{2}(f(x_0)+f(x_0+h)) \\ - \frac{h^3}{4}f''(\xi_0) + \frac{1}{2}\int_{x_0}^{x_0+h} f''(\xi_x)(x-x_0)^2\,\mathrm{d}x.$$

Since f'' is bounded on $[x_0, x_0+h]$, the integral on the right is $\mathcal{O}(h^3)$. Therefore, the error term on the right is $\mathcal{O}(h^3)$ as desired. □

Higher-order quadrature rules can be derived by using higher degree Lagrange interpolants.

Example 6.3 (Simpson's rule). Let $x_i = x_0 + ih$ with $h > 0$ and $i = 0, 1, 2$. Using the Lagrange interpolating polynomial for these points:

$$f(x) = f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + \\ f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} + \frac{f^{(3)}(\xi_x)}{6}(x-x_0)(x-x_1)(x-x_2).$$

By integrating over $[x_0, x_2]$ and substituting for h, we arrive at Simpson's rule:

$$\int_{x_0}^{x_0+2h} f(x)\,\mathrm{d}x = \frac{h}{3}\bigg(f(x_0) + 4f(x_0+h) + f(x_0+2h)\bigg) + \mathcal{O}(h^4). \tag{28}$$

□

In general, for $n+1$ evenly spaced points $x_i = x_0 + ih$ with $i = 0, \ldots, n$, a quadrature rule for integrating smooth functions on $[x_0, x_n]$ is given by

$$\int_{x_0}^{x_0+nh} f(x)\,\mathrm{d}x = \sum_{j=0}^{n} w_{j,h} f(x_0+jh) + \mathcal{O}(h^{n+1}). \tag{29}$$

Here, the $\mathcal{O}(h^{n+1})$ term depends on f having a bounded $(n+1)$st derivative on $[x_0, x_n]$, and the weights

$w_{j,j}$ are

$$w_{j,h} := \int_{x_0}^{x_0+nh} L_j(x)\mathrm{d}x, \tag{30}$$

where L_j is the jth Lagrange polynomial of degree n associated to the points x_j.

6.2 Composite numerical integration

The error for the quadrature rules from the previous subsection scales with the length of the interval h that we integrate over. Larger intervals $[a, b]$ lead to larger errors. To deal with this, we can first subdivide the interval $[a, b]$ that we integrate over into equal subintervals $a = z_0 < z_1 < \cdots < z_m = b$, where $z_i = a + ih$, $h = (b - a)/m$. Then, on each subinterval $[z_i, z_{i+1}]$ of length h, we use a quadrature rule with $n + 1$ points $x_{ij} = z_i + jh/n$ for $j = 0, \ldots, n$.

Example 6.4 (Composite trapezoid rule)**.** Integrating a smooth function $f(x)$ on $[a, b]$ using the trapezoid rule on 2 subintervals:

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x &= \int_a^{a+h} f(x)\,\mathrm{d}x + \int_{a+h}^{a+2h} f(x)\,\mathrm{d}x \\ &= \frac{h}{2}\Big(f(a) + f(a+h)\Big) + \frac{h}{2}\Big(f(a+h) + f(a+2h)\Big) \\ &\quad + \mathcal{O}(h^3) \\ &= \frac{h}{2}\Big(f(a) + 2f(a+h) + f(a+2h)\Big) + \mathcal{O}\left(h^3\right).\end{aligned}$$

In comparison, for the standard trapezoid rule on the interval $[a, b] = [a, a + 2h]$:

$$\int_a^b f(x)\,\mathrm{d}x = h(f(a) + f(a+2h)) + \mathcal{O}((2h)^3),$$

which yields an error that is 8 times larger.

In general, for m equally spaced subintervals, the composite trapezoid rule over $[a, b] = [a, a+mh]$ reads

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{2}\left(f(a) + 2\sum_{j=1}^{m-1} f(a+jh) + f(b)\right) + \mathcal{O}(h^3),$$

with an error m^3 times smaller than the standard trapezoid rule on $[a, b]$. □

7 Lecture 7

7.1 Gaussian quadrature

In the previous sections, we developed quadrature rules of the form

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} w_i f(x_i),$$

where $x_i := a + i(b-a)/n$ and the w_i are chosen such that the formula is exact for polynomials up to degree n. By using a different set of n points and weights, we can achieve a quadrature rule that is accurate for polynomials up to degree $2n+1$. This is known as Gaussian quadrature.

Example 7.1 (Gauss quadrature for $n = 1$)**.** On the interval $[-1, 1]$, we seek points x_0, x_1 and weights w_0, w_1 such that the quadrature rule above is exact for polynomials up to degree 3. Inserting $a = -1$, $b = 1$, $f(x) = 1$, $f(x) = x$, $f(x) = x^2$, and $f(x) = x^3$ above yields

$$\begin{cases} 2 = \int_{-1}^{1} 1\,dx = w_0 + w_1, \\ 0 = \int_{-1}^{1} x\,dx = w_0x_0 + w_1x_1, \\ 2/3 = \int_{-1}^{1} x^2\,dx = w_0x_0^2 + w_1x_1^2, \\ 0 = \int_{-1}^{1} x^3\,dx = w_0x_0^3 + w_1x_1^3. \end{cases}$$

This is a nonlinear system of equations in 4 unknowns w_0, w_1, x_0, x_1. We can solve it by hand as follows.

From the second equation, we have $w_0x_0 = -w_1x_1$. Inserting this in to the 4th equation implies

$$w_0x_0(x_0^2 - x_1^2) = 0.$$

If $w_0 = 0$, then the second equation implies that $w_1x_1 = 0$, which then contradicts the third equation, so $w_0 \neq 0$. If $x_0 = 0$, then we reach the same contradiction, so $x_0 \neq 0$. Therefore, $x_0^2 = x_1^2$, which means that $x_0 = x_1$ or $x_0 = -x_1$. If $x_0 = x_1$, the second equation implies $w_0 = -w_1$, which contradicts the first. Therefore, $x_0 = -x_1$, so the second equation implies that $w_0 = w_1$. Then, from the first equation, we conclude that $w_0 = w_1 = 1$. The third equation then reduces to

$$2/3 = 2x_0^2,$$

so $x_0 = \pm 1/\sqrt{3}$. We choose $x_0 = -1/\sqrt{3}$, so that $x_1 = 1/\sqrt{3}$. In conclusion, the following weights and points give exact quadrature up to degree 3:

$$w_0 = w_1 = 1,\ x_0 = -1/\sqrt{3},\ x_1 = 1/\sqrt{3}.$$

□

The procedure from the previous example does not scale to $n > 1$, so we need a different way to compute the weights and points. The Legendre polynomials help with this.

Definition 7.2 (Legendre polynomials)**.** The Legendre polynomials are the family of polynomials $\{p_n\}_{n\geq 0}$ on $[-1,1]$ characterized by the following properties:

1. For all n, p_n is a degree n polynomial such that $p_n(1) = 1$.
2. For all $m \neq n$,
$$\int_{-1}^{1} p_m(x)p_n(x)\,dx = 0.$$

Example 7.3 (Legendre polynomials)**.** The first few Legendre polynomials are as follows:

$$\begin{cases} p_0(x) = 1, \\ p_1(x) = x, \\ p_2(x) = \frac{1}{2}(3x^2 - 1). \end{cases}$$

□

Proposition 7.4. *The first $n+1$ Legendre polynomials are a basis for polynomials of degree at most n.*

Proof. Suppose that
$$\sum_{i=0}^{n} c_i p_i = 0.$$
Then, by multiplying by p_j and integrating over $[-1,1]$, we conclude that
$$c_j \int_{-1}^{1} p_j(x)^2\,dx = 0.$$
Since $p_j(1) = 1$, $p_j \neq 0$, so we must have that $c_j = 0$. Since j is arbitrary, $\{p_0, \ldots, p_n\}$ is a linearly independent set of $n+1$ polynomials of degree at most n. Since the dimension of this space is equal to the number of Legendre polynomials in the set, we are done. □

Given $p_0, \ldots, p_{n-1}$, one can compute p_n by expanding it in the following basis:
$$p_n(x) = \sum_{i=0}^{n-1} a_i p_i(x) + a_n x^n$$
and obtaining an $(n+1)\times(n+1)$ system of linear equations by requiring
$$\begin{cases} \int_{-1}^{1} p_j(x)p_n(x)\,dx = 0 \text{ for all } 0 \leq j \leq n-1, \\ p_n(1) = 1. \end{cases}$$

Example 7.5. Given p_0, p_1, and p_2 above, we compute p_3 as
$$p_3(x) = a_0p_0(x) + a_1p_1(x) + a_2p_x(x) + a_3x^3.$$

Now, the constraints read

$$\begin{cases} 0 = \int_{-1}^{1} p_0(x)p_3(x)\,dx = a_0 \int_{-1}^{1} p_0(x)^2\,dx, \\ 0 = \int_{-1}^{1} p_1(x)p_3(x)\,dx = a_1 \int_{-1}^{1} p_1(x)^2\,dx + a_3 \int_{-1}^{1} p_1(x)x^3\,dx, \\ 0 = \int_{-1}^{1} p_2(x)p_3(x)\,dx = a_2 \int_{-1}^{1} p_2(x)^2\,dx \\ 1 = p_3(1) = a_0 + a_1 + a_2 + a_3. \end{cases}$$

Thus, we see that $a_0 = 0$, $a_2 = 0$, so we have the following linear system for a_1 and a_3:

$$\begin{cases} 2/3a_1 + 2/5a_3 = 0, \\ \qquad a_1 + a_3 = 1. \end{cases}$$

This can be easily solved, so we conclude that $p_3(x) = a_1p_1(x) + a_3x^3$. □

There exists a general formula for p_n, but its derivation is beyond the scope of these notes. Here's how they relate to Gaussian quadrature. We will not cover the proofs.

Lemma 7.6 (Roots of Legendre polynomials)**.** *Let p_n be the nth Legendre polynomial. Then, p_n has n distinct roots in the interval $(-1, 1)$.*

Theorem 7.7 (Gaussian quadrature via Legendre polynomials)**.** *For $n \geq 0$, let $x_0, \ldots, x_n$ be the roots of the $n + 1$ Legendre polynomial p_{n+1}. Let*

$$w_i := \frac{2}{(1 - x_i^2)p'_{n+1}(x_i)^2}.$$

Then, the Gauss quadrature rule

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

is exact for polynomials up to degree $2n + 1$.

There are tabulations of Gauss quadrature weights and points on $[-1, 1]$ online. To obtain the corresponding quadrature rule on an arbitrary interval $[a, b]$, we can use the following change of variables:

$$x = \frac{a+b}{2} + \frac{b-a}{2}\widehat{x}, \qquad dx = \frac{b-a}{2}\,d\widehat{x}.$$

That is,

$$\begin{aligned} \int_a^b f(x)\,dx &= \int_{-1}^{1} f\left(\frac{a+b}{2} + \frac{b-a}{2}\widehat{x}\right)\frac{b-a}{2}\,d\widehat{x} \\ &\approx \sum_{i=0}^{n} \frac{b-a}{2}\widehat{w}_i f\left(\frac{a+b}{2} + \frac{b-a}{2}\widehat{x}_i\right) \\ &= \sum_{i=0}^{n} w_i f(x_i), \end{aligned}$$

where $\widehat{w}_i$ and $\widehat{x}_i$ are the Gauss quadrature weights and points on the interval $[-1,1]$ and the corresponding weights and points on $[a,b]$ are

$$w_i = \frac{b-a}{2}\widehat{w}_i, \qquad x_i = \frac{a+b}{2} + \frac{b-a}{2}\widehat{x}_i.$$

7.2 Euler's method for ordinary differential equations (ODEs)

We now wish to develop numerical methods for solving ODE problems of the form

$$y'(t) = f(t, y(t)), \qquad y(0) = y_0, \tag{31}$$

where $f(t,y)$ is a given function, y_0 is a given initial condition, and the unknown is $y(t)$. We recall some ODE theory concerning existence and uniqueness of solutions to the above problem.

Definition 7.8 (Lipschitz)**.** A function of two real variables $f(t,y)$ is Lipschitz with respect to y if there is a constant $C > 0$ such that for all t, y_1, y_2,

$$|f(t,y_1) - f(t,y_2)| \le C|y_1 - y_2|.$$

We call C a Lipschitz constant of f.

Proposition 7.9. *If $f(t,y)$ is Lipschitz with respect to y, it is uniformly continuous with respect to y.*

Proof. Let $\varepsilon > 0$, and let $C > 0$ be a Lipschitz constant of f. Then, for each t, y_0, when $|y - y_0| < \delta := \varepsilon/C$,

$$|f(t,y) - f(t,y_0)| \le C|y - y_0| < \varepsilon.$$

□

Proposition 7.10. *If $f(t,y)$ is differentiable with respect to y and there is a constant C such that*

$$|\partial_y f(t,y)| \le C$$

for all t, y, then f is uniformly Lipschitz with respect to y.

Proof. Fix t, y_1, y_2. Applying the Mean Value Theorem in y, there is some $y_3 \in (y_1, y_2)$ such that

$$|f(t,y_1) - f(t,y_2)| = |\partial_y f(t,y_3)||y_1 - y_2| \le C|y_1 - y_2|.$$

Since C is independent of t, y_1, and y_2, we are done. □

Example 7.11. The function $f(t,y) = t - y^2 + 1$ is Lipschitz in y, but $f(t,y) = \sqrt{y}$ is not.

Theorem 7.12 (Picard–Lindelöf)**.** *If f is continuous with respect to t and uniformly Lipschitz with respect to y, then the ODE problem* (31) *has a unique solution.*

If a problem does not have a solution at all, it makes no sense to develop numerical methods for it. If a problem has a solution, but it is not unique, then additional information must be supplied to choose a unique solution to approximate. There is one other additional property that determines how feasible it is to approximate a solution to an ODE problem. We discuss this now.

Definition 7.13 (Stability)**.** Consider the ODE problem (31) and the following slightly perturbed problem:

$$z'(t) = f(t, z(t)) + \varepsilon, \qquad z(0) = y_0 + \delta.$$

Suppose that $z(t)$ and $y(t)$ belong to a space of function with a norm $\|\cdot\|$ defined on it. We say that (31) is stable if, whenever ε and δ are small, the difference $\|y - z\|$ is small.

That is, small perturbations in the function f or the initial data y_0 do not give vastly different solutions to the ODE. If the ODE problem has a unique solution and is stable, we say the problem is well-posed. Well-posed problems are much easier to solve than ill-posed ones. We will only concern ourselves with well-posed problems in this course.

Now, assuming we have a well-posed problem, how do we numerically compute the solution to (31)? A simple idea is to replace the derivatives in (31) with finite differences that we learned from a previous section. By using forward differences, we obtain the forward Euler method, also known as the explicit Euler method.

Suppose we want to solve the ODE problem from $t = 0$ until some final time t_F. We let $h = t_F/N$ denote the timestep size, and we set $t_n = nh$. A forward difference of $y'(t)$ at $t = t_n$ reads

$$y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{h}.$$

Motivated by this, we consider the following algebraic procedure. We start with the initial condition y_0. Then, for $n \geq 0$, after obtaining y_n, to obtain y_{n+1}, we solve

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

Or, after rearranging, we get the following explicit Euler update:

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{32}$$

Whether the computed values y_n accurately approximate $y(t_n)$ depends on the function f and the step size h. Let

$$e_n := y(t_n) - y_n$$

denote the error at time t_n. Assuming that the exact solution y is smooth,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\xi_n)$$

for some $\xi_n \in (t_n, t_{n+1})$. Using the ODE:

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}y''(\xi_n).$$

Subtracting (32) from this equation:

$$e_{n+1} = e_n + h\left(f(t_n, y(t_n)) - f(t_n, y_n)\right) + \frac{h^2}{2}y''(\xi_n).$$

Now, assume that f is Lipschitz in y with constant $L > 0$. Then, by taking absolute values and applying the triangle inequality with the Lipschitz assumption,

$$|e_{n+1}| \leq (1+hL)|e_n| + \frac{h^2}{2}|y''(\xi_n)|.$$

Suppose that y'' is bounded with constant $M > 0$. Then, by recursively applying this inequality:

$$\begin{aligned}
|e_{n+1}| &\leq (1+hL)\left((1+hL)|e_{n-1}| + \frac{h^2}{2}M\right) + \frac{h^2}{2}M \\
&= (1+hL)^2|e_{n-1}| + \frac{h^2}{2}M(1+(1+hL)) \\
&\vdots \\
&\leq (1+hL)^{n+1}|e_0| + \frac{h^2}{2}M\sum_{j=0}^{n}(1+hL)^j.
\end{aligned}$$

From the initial condition, $e_0 = 0$, so

$$|e_{n+1}| \leq \frac{h^2}{2}M\sum_{j=0}^{n}(1+hL)^j.$$

Recalling the geometric sum identity,

$$\sum_{j=0}^{n} r^j = \frac{1-r^{n+1}}{1-r},$$

we apply this to the inequality above:

$$|e_{n+1}| \leq \frac{h^2}{2}M\frac{(1+hL)^{n+1}-1}{hL} = \frac{h}{2}\frac{M}{L}\left((1+hL)^{n+1}-1\right).$$

We now recall the following property of the exponential function:

$$e^x = \lim_{m\to\infty}\left(1+\frac{x}{m}\right)^m,$$

and, when $x > 0$,

$$e^x \geq \left(1+\frac{x}{m}\right)^m$$

for all m. Using this above for $m = n+1$ and $x = (n+1)hL = t_{n+1}L$, we conclude the following.

Theorem 7.14 (Accuracy of the forward Euler method)**.** *Suppose $f(t,y)$ is Lipschitz in y with constant $L > 0$, and suppose that the solution y to* (31) *has a bounded second derivative with bound $M > 0$. Then, the forward Euler method* (32) *with timestep $h > 0$ and discrete time points $t_n = nh$ has the following error bound for all n:*

$$|y(t_n) - y_n| \leq h\frac{M}{2L}\left(e^{t_nL} - 1\right).$$

Thus, the forward Euler method is only first-order accurate.

8 Lecture 8

8.1 Higher-order Taylor methods

In the previous section, we derived the forward Euler method by using a forward difference approximation to the derivative. We also could have derived the method by using Taylor expansions and replacing certain derivatives with the function $f(t, y)$ from the ODE. Let us do this now, since we can generalize it to higher-order accurate methods later.

Let $y(t)$ be the solution to the ODE (31), and assume that y is smooth. Then,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\xi_n)$$

for some $\xi_n \in (t_n, t_{n+1})$. Using the ODE (31),

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}y''(\xi_n).$$

By dropping the $\mathcal{O}(h^2)$ term, we obtain the forward Euler method:

$$y_{n+1} = y_n + hf(t_n, y_n).$$

If we continue further out in the Taylor expansion, we can obtain higher-order accurate methods. For example, expanding out one more term:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(\xi_n).$$

We replace $y'(t_n)$ by $f(t_n, y(t_n))$. To deal with $y''(t_n)$, we take a derivative of (31):

$$\begin{aligned} y''(t) &= \partial_t f(t, y(t)) + \partial_y f(t, y(t))y'(t) \\ &= \partial_t f(t, y(t)) + \partial_y f(t, y(t))f(t, y(t)). \end{aligned}$$

Inserting this and dropping the $\mathcal{O}(h^3)$ term, we obtain a second-order accurate explicit method:

$$y_{n+1} = y_n + hf(t_n, y_n) + \frac{h^2}{2}\bigg(\partial_t f(t_n, y_n) + \partial_y f(t_n, y_n)f(t_n, y_n)\bigg). \tag{33}$$

Now, we need f, $\partial_t f$, and $\partial_y f$ to be Lipschitz in y in order to prove an error bound similar to the forward Euler method.

Example 8.1 (Second-order accurate explicit method)**.** Let $f(t, y) = y^3$. Then the second-order accurate scheme above reads

$$y_{n+1} = y_n + hy_n^3 + \frac{3h^2}{2}y_n^5.$$

□

8.2 Runge–Kutta methods

Computing derivatives of $f(t, y(t))$ is tedious and restricts the methods of the previous section to only be useful for sufficiently smooth f. To circumvent this, we can seek to replace derivatives of $f(t, y(t))$ with values of the form $f(t+\alpha, y+\beta)$. This process leads to a class of methods called Runge–Kutta methods. We explain with an example.

Returning to the second-order explicit scheme (33), if we expand $f(t+\alpha, y+\beta)$ in a Taylor expansion in both variables:

$$f(t+\alpha, y+\beta) = f(t,y) + \alpha\partial_t f(t,y) + \beta\partial_y f(t,y) + \mathcal{O}(\alpha^2) + \mathcal{O}(\alpha\beta) + \mathcal{O}(\beta^2).$$

If we set $\alpha = h/2$ and $\beta = hf(t,y)/2$,

$$f(t + h/2, y + hf(t,y)/2) = f(t,y) + \frac{h}{2}\left(\partial_t f(t,y) + \partial_y f(t,y) f(t,y)\right) + \mathcal{O}(h^2).$$

Dropping the $\mathcal{O}(h^2)$ term, we see that the term on the right matches up with the terms in (33). Therefore, by making the substitution above, we obtain the following second-order explicit Runge–Kutta scheme:

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right). \tag{34}$$

For even higher-order methods, the procedure is similar. Perform a high-order Taylor expansion of $y(t_{n+1})$, replacing derivatives of y with partial derivatives of f. Then, for the partial derivatives of f, replace them with suitable values of the form $f(t+\alpha, y+\beta)$ where α and β come from matching coefficients with higher-order Taylor expansions of f. A popular high-order explicit Runge–Kutta scheme is the 4th-order Runge–Kutta scheme, also known as RK4:

$$\begin{cases} k_1 := f(t_n, y_n), \\ k_2 := f(t_n + h/2, y_n + h/2k_1), \\ k_3 := f(t_n + h/2, y_n + h/2k_2), \\ k_4 := f(t_n + h, y_n + hk_3), \\ y_{n+1} = y_n + \dfrac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \end{cases} \tag{35}$$

9 Lecture 9

9.1 Multistep methods

Consider the ODE

$$y'(t) = f(t, y(t)) \tag{36}$$

on an interval $a \le t \le b$ with initial condition $y(0) = \alpha$. We subdivide the interval $[a, b]$ into discrete time points $t_i = a + ih$, where $h = (b - a)/N$. Then, we integrate the ODE on subinterval $[t_i, t_{i+1}]$:

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

The idea of multistep methods is to replace the integrand $\phi(t) := f(t, y(t))$ above with a Lagrange interpolant.

Example 9.1 (2-step explicit Adams–Bashforth method)**.** Consider the Lagrange interpolant to ϕ at the points t_i and t_{i-1}:

$$p(t) = \phi(t_{i-1})\frac{t - t_i}{t_{i-1} - t_i} + \phi(t_i)\frac{t - t_{i-1}}{t_i - t_{i-1}}.$$

Replacing ϕ with p leads to the 2-step explicit Adams–Bashforth method:

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

Given two initial conditions y_0, y_1 at times t_0, t_1, for all $i \ge 1$, the equation above gives an explicit update y_{n+1} constructed from the solution at the previous 2 timesteps y_i and y_{i-1}. Since we use an equal step size h, $t_i - t_{i-1} = h$ for all i, so the integrals can be explicitly computed to get the coefficients. It is convenient to make change of variables $t = t_i + sh$ with $0 \le s \le 1$. Doing this in the first integral yields

$$\int_{t_i}^{t_{i+1}} \frac{t - t_i}{t_{i-1} - t_i}\, dt = \int_0^1 \frac{sh}{-h} h\, ds = -\frac{h}{2}.$$

For the second integral,

$$\int_{t_i}^{t_{i+1}} \frac{t - t_{i-1}}{t_i - t_{i-1}}\, dt = \int_0^1 \frac{h(1+s)}{h} h\, ds = \frac{3h}{2}.$$

Therefore, the method is

$$y_{i+1} = y_i - \frac{h}{2} f(t_i, y_{i-1}) + \frac{3h}{2} f(t_i, y_i). \tag{37}$$

□

By using the points t_i, and t_{i-1}, we constructed an explicit method. If we now include the point t_{i+1} as well, we get a family of implicit methods.

Example 9.2 (Implicit method)**.** We now replace ϕ by the Lagrange interpolant at the points t_{i+1} and t_i:

$$p(t) = \phi(t_i)\frac{t - t_{i+1}}{t_i - t_{i+1}} + \phi(t_{i+1})\frac{t - t_i}{t_{i+1} - t_i}.$$

Then, by changing variables $t = t_i + sh$ for $0 \le s \le 1$,

$$\begin{aligned}\int_{t_i}^{t_{i+1}} p(t)\,dt &= f(t_i, y_i)\int_0^1 \frac{h(s-1)}{-h} h\,ds + f(t_{i+1}, y_{i+1})\int_0^1 \frac{hs}{h} h\,ds \\ &= \frac{h}{2}\bigg(f(t_i, y_i) + f(t_{i+1}, y_{i+1})\bigg).\end{aligned}$$

Thus, we arrive at the Crank–Nicolson method:

$$y_{i+1} = y_i + \frac{h}{2}\bigg(f(t_i, y_i) + f(t_{i+1}, y_{i+1})\bigg).$$

Strictly speaking, this is a one-step method, since it only involves 2 timesteps t_i and t_{i+1}. Since we must solve a (not necessarily linear) equation for y_{i+1}, this is an implicit method. We include it here since it is derived in the same way as the explicit multistep method from the previous example. □

Definition 9.3 (Multistep method)**.** A multistep method with m steps is a method of the form

$$w_{i+1} = a_{m-1}w_i + \cdots + a_0 w_{i+1-m} + h\Big\{b_m f(t_{i+1}, w_{i+1}) + \cdots + b_0 f(t_{i+1-m}, w_{i+1-m})\Big\},$$

where the coefficients a_j and b_j are given. □

If $b_m = 0$, the method is an explicit method, known as an Adams–Bashforth method. Otherwise, the method is implicit, known as an Adams–Moulton method. To solve for w_{i+1}, the previous m timesteps $w_i, \dots, w_{i+1-m}$ must be already computed. The definition allows for $m = 1$, so multistep methods are generalizations of one-step methods we discussed in the previous sections.

Example 9.4 (Explicit 4-step Adams–Bashforth)**.** To derive the explicit 4-step Adams–Bashforth method, we use the Lagrange interpolating polynomial p at the points $t_i, t_{i-1}, t_{i-2}, t_{i-3}$. Recalling that $t_j - t_k = (j-k)h$, we have

$$\begin{aligned} p(t) &= f(t_i, y_i)L_i(t) + f(t_{i-1}, y_{i-1})L_{i-1}(t) + f(t_{i-2}, y_{i-2})L_{i-2}(t) + f(t_{i-3}, y_{i-3})L_{i-3}(t), \\ L_i(t) &= \frac{(t - t_{i-1})(t - t_{i-2})(t - t_{i-3})}{6h^3}, \\ L_{i-1}(t) &= \frac{(t - t_i)(t - t_{i-2})(t - t_{i-3})}{-2h^3}, \\ L_{i-2}(t) &= \frac{(t - t_{i-1})(t - t_i)(t - t_{i-3})}{2h^3}, \\ L_{i-3}(t) &= \frac{(t - t_{i-1})(t - t_{i-2})(t - t_i)}{-6h^3}. \end{aligned}$$

Now, since $t = t_i + sh$,

$$\int_{t_i}^{t_{i+1}} L_i(t)\,dt = \frac{h}{6}\int_0^1 (s+1)(s+2)(s+3)\,ds,$$
$$\int_{t_i}^{t_{i+1}} L_{i-1}(t)\,dt = -\frac{h}{2}\int_0^1 s(s+2)(s+3)\,ds,$$
$$\int_{t_i}^{t_{i+1}} L_{i-2}(t)\,dt = \frac{h}{2}\int_0^1 s(s+1)(s+3)\,ds,$$
$$\int_{t_i}^{t_{i+1}} L_{i-3}(t)\,dt = -\frac{h}{6}\int_0^1 s(s+1)(s+2)\,ds.$$

All of these integrals are of the form

$$\int_0^1 (s+i)(s+j)(s+k)\,ds = \frac{1}{4} + \frac{i+j+k}{3} + \frac{ij+ik+jk}{2} + ijk.$$

Using this, the coefficients are

$$\int_{t_i}^{t_{i+1}} L_i(t)\,dt = \frac{55h}{24},$$
$$\int_{t_i}^{t_{i+1}} L_{i-1}(t)\,dt = -\frac{59h}{24},$$
$$\int_{t_i}^{t_{i+1}} L_{i-2}(t)\,dt = \frac{37h}{24},$$
$$\int_{t_i}^{t_{i+1}} L_{i-3}(t)\,dt = -\frac{9h}{24}.$$

Thus, the method is

$$y_{i+1} = y_i + \frac{h}{24}\Big(55f(t_i,y_i) - 59f(t_{i-1},y_{i-1}) + 37f(t_{i-2},y_{i-2}) - 9f(t_{i-3},y_{i-3})\Big). \tag{38}$$

□

9.2 Predictor-corrector methods

As we saw from a previous example, implicit multistep methods require solving a possibly nonlinear algebraic equation for w_{n+1}. While one could use Newton's method at each iteration to solve the nonlinear equation, another approach is to use a predictor-corrector method. The idea is best explained with an example.

Example 9.5 (Predictor-corrector)**.** Consider the Crank–Nicolson scheme:

$$w_{i+1} = w_i + \frac{h}{2}\Big(f(t_i,w_i) + f(t_i,w_{i+1})\Big).$$

If f is linear in w, then this equation can be explicitly solved for w_{i+1}. For example, if $f(t,w) = w$, then

$$w_{i+1} = \frac{1}{1-h/2}\left(1+\frac{h}{2}\right)w_i.$$

On the other hand, if f is not linear in w, then the equation cannot be solved by hand. For example, if $f(t,w) = \sin(w)$:

$$w_{i+1} = w_i + \frac{h}{2}\bigg(\sin(w_i) + \sin(w_{i+1}) \bigg).$$

The idea of a predictor-corrector method is to replace w_{i+1} on the right-hand side by a predictor $\widetilde{w}_{i+1}$ that arises from an explicit method:

$$w_{i+1} = w_i + \frac{h}{2}\bigg(f(t_i, w_i) + f(t_i, \widetilde{w}_{i+1}) \bigg).$$

As a simple example, $\widetilde{w}_{i+1}$ could be computed with an explicit Euler update:

$$\widetilde{w}_{i+1} = w_i + hf(t_i, w_i).$$

□

In general, for a predictor-corrector method, replace w_{i+1} in the right-hand side of an implicit method by a predictor $\widetilde{w}_{i+1}$ obtained through an explicit method.

9.3 Convergence and stability and of one-step methods

We now discuss a general theory that dictates when a one-step method converges to the solution of the model ODE problem (36). We also discuss the related notions of consistency and stability. Throughout this section, we consider a general one-step method of the form

$$w_{i+1} = w_i + h\Phi(t_i, w_i, h), \tag{39}$$

where $\Phi(t,w,h)$ is a given function.

Definition 9.6 (Local truncation error)**.** The local truncation error (LTE) of (39) is the difference

$$\tau_{i+1}(h) := \frac{y(t_i + h) - y(t_i)}{h} - \Phi(t_i, y(t_i), h),$$

where $y(t)$ is the exact solution to (36).

The LTE is the error that comes from replacing w_i with $y(t_i)$ in (39).

Example 9.7 (LTE of the forward Euler method)**.** For the forward Euler method:

$$w_{i+1} = w_i + hf(t_i, w_i),$$

$\Phi(t_i, w_i, h) = f(t_i, w_i)$, so the LTE is

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

Since y is smooth, doing a Taylor expansion about t_i yields

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

Since y solves the ODE, we replace y' by $f(t, y)$. Rearranging yields

$$\tau_{i+1}(h) = \frac{h^2}{2} y''(\xi_i)$$

for some $\xi_i \in (t_i, t_{i+1})$. □

Definition 9.8 (Consistency)**.** The one-step method (39) is consistent if the LTE satisfies

$$\lim_{h \to 0} \max_i |\tau_i(h)| = 0.$$

□

Definition 9.9 (Convergence)**.** The one-step method (39) is convergent if

$$\lim_{h \to 0} \max_i |w_i - y(t_i)| = 0,$$

where the w_i come from (39) and $y(t)$ is the exact solution of (36). □

Definition 9.10 (Stability)**.** The one-step method (39) is stable if there is a constant $K > 0$ such that

$$|u_i - v_i| \leq K|u_0 - v_0| \text{ for all } i$$

whenever $\{u_i\}_{i \geq 1}$ and $\{v_i\}_{i \geq 1}$ satisfy the difference equation (39). □

The type of stability considered in the previous definition is that small changes in the initial conditions should lead to only small changes in the solution.

Theorem 9.11 (Convergence of one-step methods)**.** *Suppose that there is $h_0 > 0$ such that $\Phi(t, w, h)$ satisfies a Lipschitz condition in w with constant $L > 0$ on the set*

$$D := \{(t, w, h) \mid a \leq t \leq b,\ -\infty < w < \infty,\ 0 \leq h \leq h_0\},$$

i.e.

$$|\Phi(t, w_1, h) - \Phi(t, w_2, h)| \leq L|w_1 - w_2|$$

for all (t, w_1, h), $(t, w_2, h) \in D$. Then,

1. *The method is stable.*
2. *The following are equivalent:*
   - *(a) The difference method converges.*
   - *(b) The difference method is consistent.*
   - *(c) $\phi(t, y(t), 0) = f(t, y(t))$ for all $a \leq t \leq b$.*
3. *Suppose there is a function $\tau(h)$ such that, for all i, the LTE satisfies $|\tau_i(h)| \leq \tau(h)$ when $0 \leq h \leq h_0$. Then*

$$|y(t_i) - w_i| \leq \frac{\tau(h)}{L} e^{L(t_i - a)}$$

for all i.

Proof. We only prove item 1. The other results are beyond the scope of these notes. Let $\{u_i\}_i$, $\{v_i\}_i$ solve the difference equation with different initial conditions u_0, v_0. Then, from the Lipschitz condition on Φ,

$$\begin{aligned}|u_{i+1} - v_{i+1}| &= |u_i + h\Phi(t_i, u_i, h) - (v_i + h\Phi(t_i, v_i, h))| \\ &\leq |u_i - v_i| + hL|u_i - v_i|.\end{aligned}$$

Iterating this inequality yields

$$|u_{i+1} - v_{i+1}| \leq (1 + hL)^{i+1}|u_0 - v_0|.$$

Thus, by setting $K = (1 + hL)^N$, where N is the number of discrete timesteps at mesh size h, we have stability. $\square$

10 Lecture 10

10.1 Convergence and stability of multistep methods

The definition of convergence 9.9 in the previous section also extends to ggeneral multistep methods of the form

$$w_{i+1} = a_{m-1}w_i + \cdots + a_0 w_{i+1-m} + hF(t_i, h, w_{i+1}, \ldots, w_{i+1-m}). \tag{40}$$

Recall that we must specify the first m starting values to initialize the method, i.e. we must set

$$w_0 = \alpha_0, \cdots, w_{m-1} = \alpha_{m-1}.$$

The local truncation error now has the following form:

$$\tau_{i+1}(h) = \frac{y(t_{i+1}) - a_{m-1}y(t_i) - \cdots - a_0 y(t_{i+1-m})}{h} - F(t_i, h, y(t_{i+1}), \ldots, y(t_{i+1-m})). \tag{41}$$

For consistency of multistep methods, we must consider the local truncation error and the inital values.

Definition 10.1 (Consistency for a multistep method)**.** The multistep method is consistent if

$$\lim_{h\to 0} \max_{i\geq m} |\tau_i(h)| = 0,$$
$$\lim_{h\to 0} \max_{i\leq m-1} |\alpha_i - y(t_i)| = 0.$$

□

For a one-step method, one usually sets $\alpha_0 = y(t_0)$, so the second condition is automatically satisfied.

For one-step methods, stability was a fairly straightforward concept. For multistep methods, there is more nuance that must be accounted for. To understand stability of multistep methods, we use the characterstic polynomial associated to (40). It is obtained by setting $F = 0$ and inserting $w_i = \lambda^i$ into the equation and factoring out λ^{i+1-m}.

Definition 10.2 (Characteristic polynomial)**.** The characteristic polynomial associated to (40) is

$$p(\lambda) := \lambda^m - a_{m-1}\lambda^{m-1} - \cdots - a_1\lambda - a_0.$$

□

The roots of this polynomial characterize the stability of the multistep method.

Definition 10.3 (Stability via the characteristic polynomial)**.** Let λ_i be the roots of the characteristic polynomial. If all $|\lambda_i| \leq 1$ and, when $|\lambda_i| = 1$, λ_i is a simple root (multiplicity 1), the difference method is said to satisfy the root condition. If the root condition is satisfied and the only root with $|\lambda_i| = 1$ is $\lambda_i = 1$, then the method is called strongly stable. If the root condition is satisfied and there are more than 1 distinct roots with magnitude 1, the method is weakly stable. If the root condition is not satisfied, then the method is unstable. □

The discussion in section 5.10 of the book gives further details on stability. The following theorem is the main result of this section.

Theorem 10.4 (Convergence and stability of multistep methods)**.** *Suppose that*

1. *If* $f = 0$*, then* $F = 0$*.*

2. *F satisfies a Lipschitz condition in the w_i's: there is a constant $L > 0$ such that*

$$|F(t_i, h, w_{i+1}, \dots, w_{i+1-m}) - F(t_i, h, \widetilde{w}_{i+1}, \dots, \widetilde{w}_{i+1-m})| \\ \leq L\left(|w_{i+1} - \widetilde{w}_{i+1}| + \cdots + |w_{i+1-m} - \widetilde{w}_{i+1-m}|\right).$$

Then,

1. *The method is stable iff it satisfies the root condition.*
2. *If the method is consistent, then the method is stable iff it is convergent.*

□

10.2 Linear stability

The notion of stability in the previous subsection is also known as zero-stability, since it involves examining the characteristic polynomial associated to the difference equation when $F = 0$. This is equivalent to studying the ODE problem when $f = 0$. Now, we consider another notion of stability, called linear stability. It has important consequences for so-called stiff differential equations. See section 5.11 in the book for further discussion about stiff ODEs. Here, we just collect the relevant results about linear stability.

For linear stability, we consider the test ODE (36) with $f(t, y) = -\lambda y$ with $\lambda < 0$. This has exact solution $y(t) = \alpha e^{\lambda t}$, which should exponentially decay. We now investigate the multistep methods of the previous sections to see when this exponential decay property holds for the discrete solution w_i.

Example 10.5 (Euler's method)**.** With forward Euler, we have

$$w_{i+1} = (1 + \lambda h)w_i = \cdots = (1 + \lambda h)^{i+1} w_0.$$

Therefore, we need $|1 + \lambda h| < 1$ in order for the solution to decay. This puts a restriction on the timestep size h. We say that the forward Euler method is conditionally stable.

For backward Euler:

$$w_{i+1} = w_i + \lambda h w_{i+1},$$

so

$$w_{i+1} = \frac{1}{1 - \lambda h} w_i = \cdots = \frac{1}{(1 - \lambda h)^{i+1}} w_0.$$

Since $\lambda < 0$ and $h > 0$, $1 - \lambda h > 1$, so w_{i+1} will always decay. We say that backward Euler is unconditionally stable. □

For multi-step methods, the algebra is more complicated.

Example 10.6 (Multi-step methods)**.** We consider the following 2-step method:

$$w_{i+1} = w_i + \frac{h}{2}(3f(t_i, w_i) - f(t_{i-1}, w_{i-1})).$$

Applying this to $f(t, y) = \lambda y$ yields

$$w_{i+1} - (1 + \frac{3\lambda h}{2})w_i + \frac{h\lambda}{2} w_{i-1} = 0.$$

This is a linear difference equation in the coefficients w_i. There is a well-known mathematical theory for how to handle such equations. Essentially, we just look at the corresponding characteristic polynomial:

$$z^2 - (1 + \frac{3\lambda h}{2})z + \frac{h\lambda}{2} = 0.$$

Its roots are computed by the quadratic formula. Denote them by z_1, z_2. For linear stability, we need $|z_1|, |z_2| < 1$. This puts constraints on the timestep size h. $\square$

In general, for the multistep method

$$w_{i+1} = a_{m-1}w_i + \cdots + a_0 w_{i+1-m} + h(b_m f(t_{i+1}, w_{i+1}) + \cdots + b_0 f(t_{i+1-m}, w_{i+1-m})),$$

applying this to $f(t, y) = \lambda y$ yields

$$(1 - h\lambda b_m)w_{i+1} - (a_{m-1} + h\lambda b_{m-1})w_i - \cdots - (a_0 + h\lambda b_0)w_{i+1-m} = 0.$$

The corresponding characteristic polynomial is

$$(1 - h\lambda b_m)z^m - (a_{m-1} + h\lambda b_{m-1})z^{m-1} - \cdots - (a_0 + h\lambda b_0) = 0.$$

The roots of this polynomial must all have magnitude less than 1 for linear stability.

10.3 Systems of ODEs

So far, we have only considered 1 ODE equation $y'(t) = f(t, y(t))$. More generally, one can consider systems of ODEs. Let $y_1(t), \ldots, y_n(t)$ be unknown functions of t, and let $f_1, \ldots, f_n$ be given functions of variables $t, y_1, \ldots, y_n$. Then we have the following linear ODE system:

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \end{pmatrix} = \begin{pmatrix} f_1(t, y_1(t), \ldots, y_n(t)) \\ \vdots \\ f_n(t, y_1(t), \ldots, y_n(t)) \end{pmatrix},$$

with initial conditions $y_1(0) = \alpha_1$, ..., $y_n(0) = \alpha_n$. To discretize this system, just use one of the methods on each equation in the system. See the book, section 5.9, for examples and further discussion.

One can also consider higher-order ODEs, i.e. ODEs with higher-order derivatives. Every high-order ODE can be turned into a system of first-order ODEs. We illustrate the idea with an example.

Example 10.7 (High-order ODE)**.** Consider the ODE

$$y''' = \beta_1 y'' + \beta_2 y$$

with initial conditions $y(0) = \alpha_0$, $y'(0) = \alpha_1$, $y''(0) = \alpha_2$. To turn this into a system of first-order ODEs, we introduce the unknowns

$$y_1 = y,\ y_2 = y' = y_1',\ y_3 = y'' = y_2'.$$

Inserting this above yields

$$y_3' = \beta_1 y_3 + \beta_2 y_1.$$

Thus, the first-order system reads

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ \beta_1 y_3 + \beta_2 y_1 \end{pmatrix},$$

with initial conditions $y_1(0) = \alpha_1$, $y_2(0) = \alpha_2$, $y_3(0) = \alpha_3$. □

In general, an nth order ODE

$$y^{(n)}(t) = F(t, y(t), y'(t), \dots, y^{(n)}(t))$$

can be turned into a system of n first-order ODEs by introduction $y_1 = y$, $y_2 = y' = y_1'$, ..., $y_n = y^{(n-1)} = y_{n-1}'$, to get the system

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ \vdots \\ y_n \\ F(t, y_1(t), y_2(t), \dots, y_n(t)) \end{pmatrix}.$$

Then, the previous techniques apply.

11 Lecture 11

11.1 Linear algebra review

A *matrix* is an $m \times n$ rectangular table of numbers, with m rows and n columns. We denote the entry in row i, column j of a matrix by $A_{i,j}$ or A_{ij}. When $m = n$, we say the matrix is a *square* matrix.

For two $m \times n$ matrices A and B, we can add them by adding each entry:

$$(A + B)_{ij} = A_{ij} + B_{ij}.$$

For a constant c, we can multiply a matrix A by multiplying each entry:

$$(cA)_{ij} = cA_{ij}.$$

For an $m \times n$ matrix A and a $n \times p$ matrix B, we can multiply them to get a new $m \times p$ matrix AB with entries

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}.$$

This matrix multiplication is only defined when the number of columns of A is the same as the number of rows of B. Thus, for arbitrary rectangular matrices A and B, the product AB may not be defined, or AB may be defined but not BA, or AB and BA may be defined but $AB \neq BA$.

In the case of square $n \times n$ matrices, AB and BA are always defined, but are not necessarily equal. In this case, we say matrix multiplication is *noncommutative*. However, matrix multiplication is *associative*: for any $n \times n$ matrices A, B, and C,

$$A(BC) = (AB)C.$$

Thus, we are free to drop parentheses when multiplying more than 2 square matrices.

The *diagonal* of a square matrix A is the collection of entries A_{ii}. A square matrix is a *diagonal matrix* if its only nonzero entries are on the diagonal. The $n \times n$ *identity matrix* I is the $n \times n$ square diagonal matrix with all 1's on the diagonal. It has the important property that

$$IA = AI = A$$

for all square $n \times n$ matrices A. In other words, I is a *multiplicative identity* for matrix multiplication.

The *determinant* of a square $n \times n$ matrix A is defined inductively on the dimension n. For $n = 1$, A has a single entry A_{11}, so

$$\det A := A_{11}.$$

For $n \geq 1$ and $1 \leq i, j \leq n$, we let M_{ij} be the $(n-1) \times (n-1)$ submatrix of A by removing the ith row and jth column. For example, with

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix},$$

$$M_{2,3} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Now, we define the determinant of A by expanding along the first row:

$$\det A := \sum_{j=1}^{n} (-1)^{1+j} A_{1j} \det M_{1j}.$$

Notice that, since M_{1j} is an $(n-1) \times (n-1)$ matrix, the definition recursively expands until we reach the base case of taking the determinant of many 1×1 matrices. Thus, this is well-defined. We expanded along the first row for concreteness, but it is a well-known theorem that the expansion can be done along any row or column. That is, for any $1 \le i \le n$,

$$\det A = \sum_{j=1}^{n} (-1)^{i+j} A_{ij} \det M_{ij},$$

and, for any $1 \le j \le n$,

$$\det A = \sum_{i=1}^{n} (-1)^{i+j} A_{ij} \det M_{ij}.$$

For a square matrix A, there is at most one square matrix A^{-1} such that

$$AA^{-1} = A^{-1}A = I.$$

When it exists, we call A^{-1} the *inverse* of A. When A^{-1} exists, we call A *invertible*. A matrix is invertible iff $\det A \neq 0$. If A and B are $n \times n$ invertible matrices, then the product AB is also invertible, and $(AB)^{-1} = B^{-1}A^{-1}$.

When a matrix A is invertible, the following procedure, called *Gaussian elimination*, can be used to compute the inverse. First, we create the augmented matrix

$$\begin{pmatrix} A & I \end{pmatrix},$$

where I is the identity matrix. Then, we perform sequences of *elementary row operations* to reduce A to a *upper-triangular* form, i.e. all entries below the diagonal of A are 0. Then, we perform *backward elimination* to reduce the left side to the identity matrix. The resulting matrix on the right side is A^{-1}. Before giving an example, we recall the basic row operations.

There are 3 types of elementary row operations:

1. Scale a row by a nonzero constant c, denoted by $R_i \to cR_i$, where R_i is the ith row.

2. Swap 2 rows, denoted by $R_i \leftrightarrow R_j$.

3. Add a multiple of one row to another row, denoted by $R_i \to R_i + cR_j$.

Now, for the example with

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}.$$

$$\begin{pmatrix}1 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 0 & 1 & 0\\ 1 & -1 & 1 & 0 & 0 & 1\end{pmatrix} \qquad R_3 \to R_3 - R_1$$

$$\begin{pmatrix}1 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 0 & 1 & 0\\ 0 & -2 & 1 & -1 & 0 & 1\end{pmatrix} \qquad R_3 \to R_3 + 2R_2$$

$$\begin{pmatrix}1 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 0 & 1 & 0\\ 0 & 0 & 1 & -1 & 2 & 1\end{pmatrix} \qquad R_1 \to R_1 - R_2$$

$$\begin{pmatrix}1 & 0 & 0 & 1 & -1 & 0\\ 0 & 1 & 0 & 0 & 1 & 0\\ 0 & 0 & 1 & -1 & 2 & 1\end{pmatrix}.$$

Thus,

$$A^{-1} = \begin{pmatrix}1 & -1 & 0\\ 0 & 1 & 0\\ -1 & 2 & 1\end{pmatrix}.$$

All of the elementary row operations correspond to multiplying by an *elementary matrix* on the left. For operation 1, the corresponding matrix is the identity matrix with the ith diagonal entry scaled by c. For operation 2, the matrix is the identity matrix with rows i and j swapped. For operation 3, the matrix is the identity matrix with c added to the (i, j) entry. For example, with 3×3 matrices,

$$(R_2 \to 2R_2) = \begin{pmatrix}1 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{pmatrix},$$

$$(R_2 \leftrightarrow R_3) = \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix},$$

$$(R_2 \leftrightarrow R_2 + 3R_1) = \begin{pmatrix}1 & 0 & 0\\ 3 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}.$$

The inverse of each elementary matrix is also an elementary matrix of the same type. For the examples above,

$$(R_2 \to 2R_2)^{-1} = (R_2 \to R_2/2) = \begin{pmatrix}1 & 0 & 0\\ 0 & 1/2 & 0\\ 0 & 0 & 1\end{pmatrix},$$

$$(R_2 \leftrightarrow R_3)^{-1} = (R_2 \leftrightarrow R_3) = \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix},$$

$$(R_2 \to R_2 + 3R_1)^{-1} = (R_2 \to R_2 - 3R_1) = \begin{pmatrix}1 & 0 & 0\\ -3 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}.$$

In general,

$$\begin{aligned}(R_i \to cR_i)^{-1} &= (R_i \to R_i/c),\\ (R_i \leftrightarrow R_j)^{-1} &= (R_i \leftrightarrow R_i/c),\\ (R_i \to R_i + cR_j)^{-1} &= (R_i \to R_i - cR_j).\end{aligned}$$

11.2 LU decompositions

A *LU decomposition* of a square matrix A is a decomposition into a product of a *lower-triangular matrix L* (all entries above the diagonal are 0), and an *upper-triangular matrix U* (all entries below the diagonal are 0):

$$A = LU.$$

Not every square matrix has a LU decomposition, and LU decompositions are not necessarily unique. If we impose that L has all 1's on its diagonal, then there is at most one such LU decomposition. The following Gaussian elimination procedure can be used to compute this unique LU decomposition of a matrix A, if it exists. We illustrate with an example to find the decomposition of

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 2 & 2 & 1 \\ 2 & 2 & 3 \end{pmatrix}.$$

We multiply on the left by elementary matrices of the 3rd kind to obtain an upper-triangular matrix. We start by using the first row to eliminate the $(2,1)$ and $(3,1)$ entries. Then, we use the second row to eliminate the $(3,2)$ entry.

$$(R_3 \to R_3 - R_1)(R_2 \to R_2 - R_1)A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 2 & 2 \end{pmatrix}.$$

$$(R_3 \to R_3 - R_2)(R_3 \to R_3 - R_1)(R_2 \to R_2 - R_1)A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

This is the matrix U we are looking for in the decomposition. Since the LU decomposition is unique, we conclude that

$$\begin{aligned} L &= ((R_3 \to R_3 - R_2)(R_3 \to R_3 - R_1)(R_2 \to R_2 - R_1))^{-1} \\ &= (R_2 \to R_2 - R_1)^{-1}(R_3 \to R_3 - R_1)^{-1}(R_3 \to R_3 - R_2)^{-1} \\ &= (R_2 \to R_2 + R_1)(R_3 \to R_3 + R_1)(R_3 \to R_3 + R_2). \end{aligned}$$

By applying both sides to the left of the identity matrix, we compute

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

12 Lecture 12

12.1 PLU decompositions

In the last example of the previous section, Gaussian elimination was possible without the need for pivoting, i.e. switching rows. When pivoting is needed, there is a slight modification to the algorithm. We illustrate again with an example to compute the decomposition of

$$A = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \\ 2 & 2 & 3 \end{pmatrix}.$$

We start by swapping rows 1 and 2:

$$(R_1 \leftrightarrow R_2)A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 2 & 2 & 3 \end{pmatrix}.$$

Now, we can proceed with the usual elimination procedure to decompose

$$(R_1 \leftrightarrow R_2)A = LU.$$

$$(R_3 \to R_3 - R_1)(R_1 \leftrightarrow R_2)A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 2 & 2 \end{pmatrix}.$$

$$(R_3 \to R_3 - R_2)(R_3 \to R_3 - R_1)(R_1 \leftrightarrow R_2)A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} = U.$$

Then,

$$\begin{aligned} L &= ((R_3 \to R_3 - R_2)(R_3 \to R_3 - R_1))^{-1} \\ &= (R_3 \to R_3 - R_1)^{-1}(R_3 \to R_3 - R_2)^{-1} \\ &= (R_3 \to R_3 + R_1)(R_3 \to R_3 + R_2) \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}. \end{aligned}$$

Thus, we have computed the LU decomposition of $(R_1 \leftrightarrow R_2)A$, so we obtain a PLU decomposition of A by

$$A = (R_1 \leftrightarrow R_2)^{-1}LU = (R_1 \leftrightarrow R_2)LU = PLU,$$

where $P = (R_1 \leftrightarrow R_2)$ is a *permutation matrix.*

12.2 Iterative methods for linear systems

To solve $Ax = b$ with iterative methods, we decompose A into

$$A = E + B,$$

where E is computationally efficient to invert or solve with, and B is the remaining part. Then, the method reads

$$Ex_{n+1} + Bx_n = b,$$

with initial guess x_0. Common choices of E are

1. Jacobi: $E = D$, the diagonal of A.
2. Gauß–Seidel: $E = D + L$, where L is the strictly lower-triangular part of A.
3. Successive over-relaxation (SOR): $E = D/\omega + L$, where $\omega > 1$ is the *relaxation parameter*.

In all cases, the remainder $B := A - E$. In general, Gauß–Seidel outperforms Jacobi, and SOR outperforms all of the methods provided that ω is chosen appropriately. Discussing how to choose ω is beyond the scope of these notes. There is a discussion in the book about relaxation techniques that explains how to choose ω. The book uses different notation from these There is also discussion online about the SOR method and other relaxation techniques.

The way we define the methods here is not how they commonly appear in the literature. The form we chose is show that all the methods are derived from decomposing $A = E + B$. The typical way that SOR is presented in the literature is by multiplying the iterative method by ω and expressing B in terms of D, L, and the strictly upper-triangular part U of A. Since $A = D + L + U$:

$$(D + \omega L)x_{n+1} + (\omega U + (\omega - 1)D)x_n = \omega b.$$

Similarly, Jacobi and Gauß–Seidel also express B in terms of D, L, and U:

$$Dx_{n+1} + (L + U)x_n = b.$$

$$(D + L)x_{n+1} + Ux_n = b.$$

12.3 Matrix norms and convergence of iterative methods

When does an iterative method converge, and how do we measure when the method is converging? To answer the first question, we consider the approximate sequence

$$Ex_{n+1} + Bx_n = b$$

and the exact solution

$$Ex + Bx = b.$$

Subtracting:

$$E(x_{n+1} - x) + B(x_n - x) = 0.$$

Rearranging:

$$x_{n+1} - x = -E^{-1}B(x_n - x).$$

Let $T := -E^{-1}B$. This is the *iteration matrix* for the method. The properties of T dictate when the iterative method converges. To proceed, we must first review *matrix norms*.

First, a *vector norm* is a function that maps an n-dimensional vector v to a positive real-number $\|v\| \geq 0$ such that

1. For all vectors v, w,
$$\|v + w\| \leq \|v\| + \|w\|$$
2. For all scalars a and vectors v,
$$\|av\| = |a|\|v\|$$
3. For all vectors v,
$$\|v\| = 0 \iff v = 0.$$

Common vector norms are the 1-norm, 2-norm, p-norm ($1 \leq p < \infty$), and ∞-norm:

$$\begin{aligned}
\|v\|_1 &:= \sum_{i=1}^{n} |v_i|, \\
\|v\|_2 &:= \sqrt{\sum_{i=1}^{n} |v_i|^2}, \\
\|v\|_p &:= \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p}, \\
\|v\|_\infty &:= \max_{i=1,\ldots,n} |v_i|.
\end{aligned}$$

Similarly, a *matrix-norm* on the space of $n \times n$ matrices is a function that maps a matrix A to a positive real number $\|A\| \geq 0$ with analogous properties to the above vector-norm properties, *and*

$$\|AB\| \leq \|A\|\|B\|$$

for all matrices A and B.

The most interesting matrix norms are ones *induced* from vector norms. If $\|\cdot\|$ is a vector norm, the corresponding matrix norm is defined as

$$\|A\| := \max_{\|v\|=1} \|Av\|,$$

where v runs over all n-dimensional vectors and A is an $n \times n$ matrix. The norms on the right are the vector norms applied to the vectors $\|v\|$ and $\|Av\|$ respectively, while the norm on the left is a matrix norm.

Proposition 12.1. *The induced norm above is indeed a matrix norm.*

Proof. Accept as fact that the norm is well-defined, i.e. the maximum is achieved and is a finite value. The proof of this is not too difficult, but still beyond the scope of these notes. We will instead verify the other properties. Since $\|(A + B)v\| \leq \|Av\| + \|Bv\|$, property 1 follows. Since $\|(aA)v\| = |a|\|Av\|$, property 2 follows.

For property 3, if $A = 0$, then $Av = 0$ for all v, so $\|Av\| = 0$ for all v. Thus, $\|A\| = 0$. Conversely, if $\|A\| = 0$, then $\|Av\| = 0$ for all v, so $Av = 0$ for all v. Thus, $A = 0$.

It remains to prove the multiplication property. For matrices A, B, and a vector v with $\|v\| = 1$, if $Bv = 0$, then $\|ABv\| = 0 \leq \|A\|\|B\|$. If $Bv \neq 0$, then let $w = Bv/\|Bv\|$, so $\|w\| = 1$ and then

$$\|ABv\|/\|Bv\| = \|Aw\| \leq \|A\|.$$

Thus, $\|ABv\| \leq \|A\|\|Bv\| \leq \|A\|\|B\|$. In all cases, $\|ABv\| \leq \|A\|\|B\|$ for all v with $\|v\| = 1$, so $\|AB\| \leq \|A\|\|B\|$. ☐

Proposition 12.2. *For any vector v, matrix A, and vector norm $\|\cdot\|$ with induced matrix norm:*

$$\|Av\| \leq \|A\|\|v\|.$$

Proof. If $v = 0$, there is nothing to prove. If $v \neq 0$, let $w = v/\|v\|$. Then $\|w\| = 1$, so

$$\|Av\|/\|v\| = \|Aw\| \leq \|A\|.$$

The result follows. ☐

Thus, for the vector-norms above, the corresponding matrix norms are

$$\begin{aligned}
\|A\|_1 &= \max_{\sum_i |v_i|=1} \sum_i \left| \sum_j A_{ij} v_j \right|, \\
\|A\|_2 &= \max_{\sum_i |v_i|^2=1} \sqrt{\sum_i \left| \sum_j A_{ij} v_j \right|^2}, \\
\|A\|_p &= \max_{\sum_i |v_i|^p=1} \left(\sum_i \left| \sum_j A_{ij} v_j \right|^p \right)^{1/p}, \\
\|A\|_\infty &= \max_{\max_i |v_i|=1} \max_i \left| \sum_j A_{ij} v_j \right|.
\end{aligned}$$

Now, these are not convenient formulas to compute with. Luckily, the following theorems fix this.

Theorem 12.3.

$$\|A\|_\infty = \max_i \sum_j |A_{ij}|.$$

Proof. For any v with $\|v\|_\infty = 1$, $|v_j| \leq 1$ for all j. Therefore, for all i

$$\left| \sum_j A_{ij} v_j \right| \leq \sum_j |A_{ij}||v_j| \leq \sum_j |A_{ij}|.$$

Thus,

$$\|A\|_\infty \leq \max_i \sum_j |A_{ij}|.$$

Now, let i_0 be such that

$$\sum_j |A_{i_0 j}| = \max_i \sum_j |A_{ij}|.$$

Let $v_j = \operatorname{sign} A_{i_0 j} \in \{1, -1\}$. Then, $\|v\|_\infty = 1$, and

$$\max_i \sum_j |A_{ij}| = \sum_j |A_{i_0 j}| = \left|\sum_j A_{i_0 j} v_j\right| \leq \max_i \left|\sum_j A_{ij} v_j\right| \leq \|A\|_\infty.$$

□

Theorem 12.4.

$$\|A\|_1 = \max_j \sum_i |A_{ij}|.$$

Proof. Let v with $\|v\| = 1$. Then,

$$\|Av\|_1 = \sum_i |\sum_j A_{ij} v_j| \leq \sum_j \sum_i |A_{ij}| v_j \leq \max_j \sum_i |A_{ij}| \sum_j |v_j| = \max_j \sum_i |A_{ij}|.$$

Now, let j_0 such that

$$\max_j \sum_i |A_{ij}| = \sum_i |A_{ij_0}|.$$

Let e_{j_0} be the j_0 standard basis vector. Then, $(Ae_{j_0})_i = A_{ij_0}$, and $\|e_{j_0}\|_1 = 1$, so

$$\max_j \sum_i |A_{ij}| = \|Ae_{j_0}\|_1 \leq \|A\|_1.$$

□

The next theorem concerning the 2-norm uses the *spectral radius* of a $n \times n$ matrix A,

$$\rho(A) := \max\{|\lambda| : \lambda \text{ eigenvalue of } A\},$$

where we recall that λ is an *eigenvalue* of A if there is a nonzero vector x, called an *eigenvector*, such that

$$Ax = \lambda x.$$

We also recall that for a matrix A, its *transpose* is the matrix A^T with entries

$$A^T_{ij} = A_{ji}.$$

Theorem 12.5.

$$\|A\|_2 = \sqrt{\rho(A^T A)}.$$

The proof is beyond the scope of these notes, as it requires the *Spectral Theorem*, which we will not cover.

Now, we can return to the iteration matrix T, where we deduced that

$$x_{n+1} - x = T(x_n - x)$$

for the iterative methods. By applying a vector norm to both sides and using the induced matrix norm:

$$\|x_{n+1} - x\| \leq \|T\| \|x_n - x\|.$$

If $\|T\| < 1$ in this induced norm, then the method will converge.