Math 151A Lecture Notes, Fall 2024

Nathan Solomon

November 8, 2024

Contents

1 Lecture 1

$f \in C([a,b])$ means that f is continuous on $[a,b]$. $f \in C^n([a,b])$ means that f is n times continuously differentiable (that is, $f^{(n)}$ is continuous) on $[a,b]$.

Intermediate Value Theorem (IVT) says that if $f \in C([a,b])$, and there exists $k \in \mathbb{R}$ such that either $f(a) \leq k \leq f(b)$ or $f(b) \leq k \leq f(a)$, then there exists at least one $c \in [a,b]$ such that $f(c) = k$. If f is strictly increasing or strictly decreasing on $[a,b]$, then c is unique.

Taylor's theorem says that if $f \in C^n([a,b])$ and $x_0 \in [a,b]$ and $f^{(n+1)}$ exists (but is not necessarily continuous) on $[a,b]$, then for every $x \in [a,b]$, there exists some $\xi_x \in [x_0, x]$ (or in $[x, x_0]$, which is technically

the same) such that $f(x) = P_n(x) + R_n(x)$, where

$$P_n(x) = \sum_{k=0}^{n} \frac{(x - x_0)^k f^{(k)}(x_0)}{k!}$$
$$R_n(x) = \frac{(x - x_0)^{n+1} f^{(n+1)}(\xi_x)}{(n+1)!}.$$

2 Lecture 2

There are 3 types of error that we need to worry about:

- **Computational error** is any error that occurs because computers can only store a finite number of digits. Rounding, truncation, overflow, and underflow are all types of computational error.
- **Truncation error** is error that occurs when you use algorithms that are designed for approximating. For example, approximating a Taylor polynomial with finitely many terms or approximating an integral with a Riemann sum will result in truncation error.
- **Data error** is error that occurs because of noise or measurement error in data. Such errors can be either random or systematic.

The floating point form $Fl(x)$, is the computer's representation of x. We can always write $Fl(x) = x + \varepsilon$, where ε is the rounding error.

When a computer does math, it will either round or chop after each step, so we want to reduce the number of Floating Point Operations (FLOPs). Rewriting an equation in "nested form" can reduce the number of FLOPs, which makes it faster and (usually) more accurate to compute. For example, calculating $x^3 - 6.x^2 + 3.2x + 1.5$ requires 8 FLOPs, but $x(x(x - 6.1) + 3.2) + 1.5$ only requires 5 FLOPs.

If p^* is an approximation of p, then the absolute error is $|p^* - p|$ and the relative error is $|p^* - p| / |p|$ (assuming $p \neq 0$).

3 Lecture 3

We say that $f(x)$ is $O(g(x))$ as $x \to +\infty$ iff there exist $M, x_0 \in \mathbb{R}$ such that $|f(x)| \leq M |g(x)|$ for any $x > x_0$. In other words, $f(x)$ is asymptotically bounded by (a scalar multiple of) $g(x)$.

We say that $f(x)$ is $O(g(x))$ as $x \to a$ iff there exist $M, \delta \in \mathbb{R}$ such that $|f(x)| \leq M |g(x)|$ whenever $|x - a| < \delta$.

For practical purposes, you can replace those less-than-or-equal-to signs with equal signs, because it would make sense to call merge sort and FFT "$O(n^2)$" algorithms when you could instead say they are "$O(n \log n)$" algorithms.

A sequence x_n is said to converge with order of convergence $\alpha \geq 1$ to x iff x_n converges to x and there exists $L \in (0, \infty)$ such that

$$\lim_{n \to \infty} \frac{|x_{n+1} - x|}{|x_n - x|^\alpha} = L.$$

L is called the asymptotic error constant.

If $\alpha = 1$ and $L < 1$, then we say x_n converges linearly. If $1 < \alpha < 2$, we say it converges super-linearly, and if $\alpha = 2$, we say it converges quadratically.

4 Lecture 4

Given a function $f \in C([a, b])$ such that $f(a)f(b) < 0$, the bisection method will converge linearly to one root of f. You start by saying $a_1 = 1, b_1 = b$, then at each step, define either a_{n+1} or b_{n+1} to be $(a_n + b_n)/2$, and leave the other endpoint unchanged. The error after n steps is less than or equal to $(b - 1)/2^n$, and the residual is $|f(p_n)|$, where p_n is our approximation of a root p

5 Lecture 5

A fixed point of a function f is a point p such that $f(p) = p$. Saying that p is a fixed point of f is equivalent to saying p is a root of the function $x \mapsto f(x) - x$.

If $f \in C([a, b])$ and $f(x) \in [a, b]$ for all $x \in [a, b]$, then f has at least one fixed point in $[a, b]$. If those criteria are both true and $f'(x)$ is defined on (a, b) and there exists $k \in (0, 1)$ such that $|f'(x)| \leq k$ for all $x \in (a, b)$, then the fixed point is unique.

Fixed point iteration (FPI) tries to approximate a fixed point p of f by taking an initial guess $p_0 \in [a, b]$ and defining $p_n = f(p_{n-1})$. If the four criteria above are all true, then the sequence p_n will converge to the unique fixed point p for any choice of $p_0 \in [a, b]$, and

$$|p_n - p| \leq k^n \max(p_0 - a, b - p_0)$$

for all n, and

$$|p_n - p| \leq \frac{k^n}{1 - k} |p_1 - p_0|$$

for all $n \geq 1$. This implies that p_n converges (at least) linearly to p.

6 Lecture 6

Newton and Rhapson's method (often just called Newton's method) converges to a root quadratically, so it can be much better than the bisection method or fixed point iteration (but it can also fail in some cases).

If $f \in C^2([a, b])$ and we want to find some root p of f, then we choose some initial guess p_0 such that $|p_0 - p|$ is small, then define

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

Newton's method is a local method, meaning it won't converge unless p_0 is sufficiently close to p. One way to get around this is to do a few iterations of a global root-finding method (like the bisection method), then switch to Newton's method.

If we don't have a computationally efficient way to calculate $f'(x)$, then instead of using Newton's method, we can use the secant method, which takes two initial guesses p_0 and p_1, then defines

$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)} \approx p_n - f(p_n) \cdot \frac{(p_n - p_{n-1})}{f(p_n) - f(p_{n-1})}.$$

Newton's method can be generalized to work for functions $f : \mathbb{R}^n \to \mathbb{R}^n$, in which case we want to find a vector p such that $f(p) = 0$. For this situation, we define

$$p_{n+1} = p_n - J_f(p_n)^{-1} f(p_n),$$

where J_f is the Jacobian matrix:

$$[J_f(x)]_{i,j} := \frac{\partial f_i(x)}{\partial x_j}.$$

7 Lecture 7

Newton's method converges quadratically ($\alpha = 2$) and the secant method converges super-linearly ($\alpha = \phi = (1 + \sqrt{5})/2 \approx 1.618$, which we will not bother to prove).

First, we want a general theorem for proving how fast FPI converges. If $g \in C^\alpha([a, b])$ for some integer $\alpha \geq 2$ and $p \in [a, b]$ is a point such that $g(p) = p$ and $g'(p) = g''(p) = \cdots = g^{(\alpha-1)}(p) = 0$, but $g^{(\alpha)} \neq 0$, then the sequence defined by $p_{n+1} = g(p_n)$ converges to p with order of convergence α for all p_0 sufficiently close to p. To prove this, write the Taylor series expansion for g centered at p, evaluate it at p_n, and apply Taylor's theorem. You will see that $L = \frac{1}{\alpha!} \lim_{n\to\infty} \left|g^{(\alpha)}(\xi_n)\right|$, where ξ_n is between p and p_n. Therefore $L = \frac{1}{\alpha!} \left|g^{(\alpha)}(p)\right|$.

If we define

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

then

$$g'(x) = x - \frac{f(x)f''(x)}{(f'(x))^2},$$

which is only defined if $f'(x) \neq 0$. If $f'(x) \neq 0$, then by the above theorem, p_n (defined by $p_{n+1} = g(p_n)$) converges to p with order of convergence $\alpha \geq 2$ for p_0 sufficiently close to p.

8 Lecture 8

A point p such that $f(p) = 0$ is called a zero (of f) of multiplicity m iff there exists a function q such that $f(x) = (x-p)^m q(x)$ for any $x \neq m$, and q is continuous in a neighborhood of m, and $q(p) \neq 0$.

The point $p \in (a, b)$ is a zero of multiplicity m of a function $f \in C^m([a, b])$ iff $0 = f(p) = f'(p) = \cdots = f^{(m-1)}(p)$, but $f^{(m)}(p) \neq 0$.

If $m \geq 1$ and p is a zero of f of multiplicity m, then the function $\mu(x) := f(x)/f'(x)$ has a zero of multiplicity 1 at p. If we want to find a root p of f using Newton's method, but we know that p is a zero (of f) of multiplicity $m > 1$, then we can instead define $p_{n+1} = p_n - \mu(p_n)/\mu'(p_n)$. This should also converge quadratically.

Atkien's Acceleration Theorem says that if p_n converges linearly to p, and $(p_{n+1}-p)(p_n-p) > 0$ for sufficiently large n, then the sequence $\hat{p}_n := p_n - (p_{n+1}-p_n)^2/(p_{n+2}-2p_{n+1}+p_n)$ satisfies $\lim_{n\to\infty} |\hat{p}_n - p| \,/\, |p_n - p| = 0$, meaning $\hat{p}_n$ converges to p faster than p_n does.

9 Lecture 9

The Weierstrass approximation theorem (also called the Stone-Weierstrass theorem) says that if $f \in C([a, b])$, then for any $\varepsilon > 0$ there exists a polynomial $P(x)$ such that $|f(x) - P(x)| < \varepsilon$ for all $x \in [a, b]$.

Given $n + 1$ data points (each data point is a pair $(x_i, f(x_i))$, and $x_i \neq x_j$ unless $i = j$), the Lagrange polynomial is the unique degree n polynomial which goes through all of those points.

Here is an explicit formula for the Lagrange polynomial of degree n:

$$P(x) := \sum_{k=0}^{n} f(x_k) L_{n,k}(x), \qquad L_{n,k}(x) := \prod_{i \in [0,n] \cap \mathbb{Z} - \{k\}} \frac{x - x_i}{x_k - x_i}.$$

Note that $\{L_{n,k}\}$ is basis for the vector space of polynomials of degree n.

10 Lecture 10

If we have $\{x_0, x_1, \ldots, x_n\} \subset [a, b]$ and $x_0 < x_1 < \cdots < x_n$, and $f \in C^{n+1}([a, b])$, and $P(x)$ is the nth degree Lagrange polynomial for f, then for all $x \in [a, b]$, there exists $\xi(x) \in [a, b]$ such that

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

11 Lecture 11

Neville's method allows you to recursively construct a Lagrange polynomial. If you have a Lagrange polynomial $P_{0,1,\ldots,k}(x)$ which interpolates through points $x_0 < x_1 < \cdots < x_k$, then for any $i \neq j$,

$$P_{0,\ldots,k}(x) = \frac{(x - x_i)P_{0,\ldots,i-1,i+1,\ldots,k}(x) - (x - x_j)P_{0,\ldots,j-1,j+1,\ldots,k}(x)}{x_j - x_i}$$

This polynomial can be constructed by filling in the lower-triangular matrix Q from left to right and top to bottom, or top to bottom and left to right:

$$Q := \begin{bmatrix} P_0(x) & 0 & 0 & \cdots \\ P_1(x) & P_{01}(x) & 0 & \cdots \\ P_2(x) & P_{12}(x) & P_{012}(x) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Q does not have to be evaluated at x, but if we want to use the Lagrange Polynomial to approximate $f(x)$, then it makes sense to calculate $P_0(x)$, then $P_{01}(x)$, then $P_{012}(x)$, and so on, until adding one more row or column doesn't change our current estimate much – that is, until $|Q_{i,i} = Q_{i+1,i+1}|$ is below our tolerance. It's good to stop before making the degree too high, because we don't want Runge phenomena – that's when the Lagrange polynomial oscillates wildly near the endpoints.

12 Lecture 12

The method of divided differences is another way to recursively calculate the LP. This method ensures it has the form

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1}).$$

We define the kth "divided difference" to be $a_k = f[x_0, x_1, \ldots, x_k]$, and define a recursive formula for those constants:

The 0th divided difference, denoted by $f[x_i]$, is defined by $f[x_i] = f(x_i)$. The first divided difference is $f[x_i, x_{i+1}] := (f[x_{i+1}] - f[x_i])/(x_{i+1} - x_i)$. The kth divided difference is

$$f[x_i, \ldots, x_{i+k}] := \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

If we want to interpolate between $n+1$ points, $\{x_0, x_1, \ldots, x_n\}$ which are equally spaced, with spacing $h := (x_n - x_0)/n$, let $x = x_0 + sh$, so we get

$$\begin{aligned} P(x) &= f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0)\cdots(x - x_{k-1}) \\ &= f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k]h^k s(s-1)\cdots(x - k + 1) \\ &= f[x_0] + \sum_{k=1}^{n} \binom{s}{k} h^k k! f[x_0, x_1, \ldots, x_k]. \end{aligned}$$

13 Lecture 13

From lecture 10, we have the theorem

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

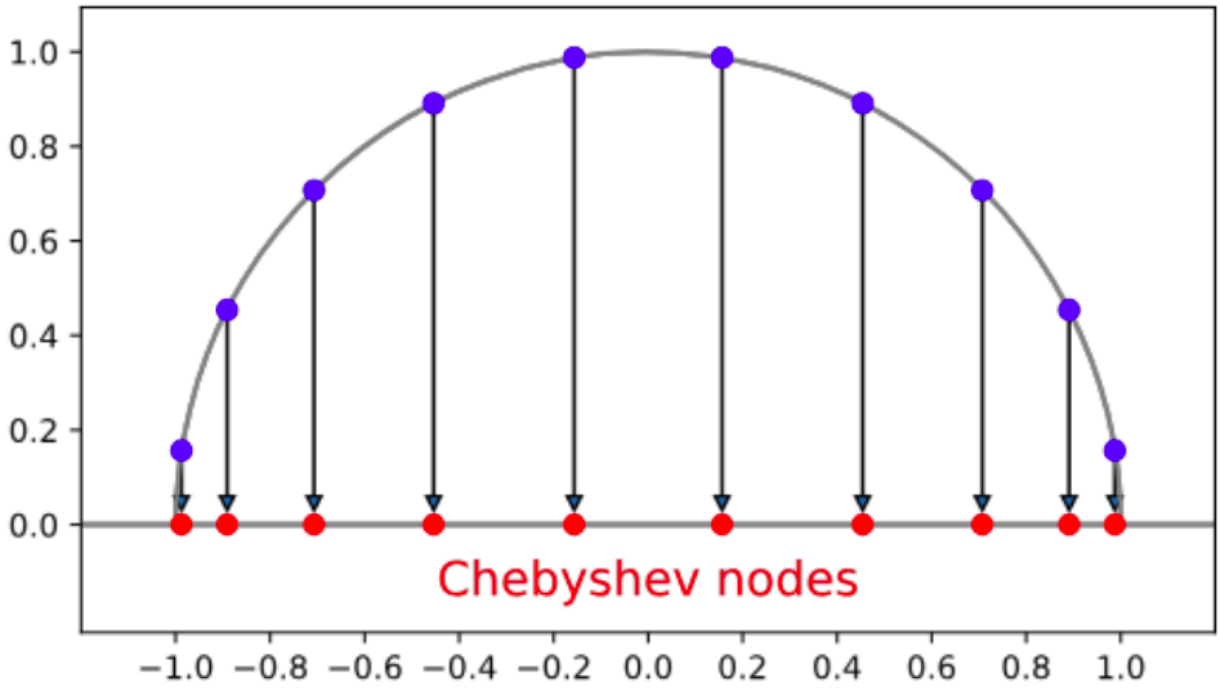
If $M \geq \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|$ and the interpolation nodes $\{x_0, x_0 + h, x_0 + 2h, \ldots, x_n\}$ are equally spaced, then

$$|f(x) - P(x)| \leq \frac{M}{(n+1)!} \cdot \max_{x\in[a,b]} \prod_{j=0}^{n} |x - x_j|.$$

But if the x_j are equally spaced, $\max_{x\in[a,b]} \prod_{j=0}^{n} |x - x_j| \leq \frac{1}{4}h^{n+1}n!$, so

$$|f(x) - P(x)| \leq \frac{Mh^{n+1}}{4(n+1)}.$$

To avoid Runge phenomena, we can replace our equally spaced interpolation nodes x_i with Chebyshev nodes, $\widetilde{x_i} := \cos\left(\frac{2i+1}{2n+2}\pi\right), i = 0, \ldots, n$ which get denser closer to the boundary of the domain we are approximating f on.



Of course, if we are interpolating between real data points, we can't choose where the nodes are.

14 Lecture 14

If you have a bunch of nonnegative integers $m_0 < m_1 < \cdots < m_n$, then the osculating polynomial approximating $f \in C^m([a,b])$ is the polynomial P of least degree such that

$$\frac{\mathrm{d}^k P(x_i)}{\mathrm{d}x^k} = \frac{\mathrm{d}^k f(x_i)}{\mathrm{d}x^k}$$

for any $i \in \{0, \ldots, n\}$ and any $k \in \{m_0, \ldots, m_n\}$. The degree of P is at most $M = n + \sum_{i=0}^{n} m_i$. If $\{m_i\} = \{0, 1\}$, then we get Hermite polynomials.

Cubic splines are a function S designed to approximate some function f with domain $[a,b]$ by interpolating through nodes $a = x_0 < x_1 < \cdots < x_n = b$.

$$S(x) = \begin{cases} S_0(x) & x_0 \leq x \leq x_1 \\ S_1(x) & x_1 \leq x \leq x_2 \\ S_{n-1}(x) & x_{n-1} \leq x \leq x_n \end{cases}$$

so on the subinterval $[x_j, x_{j+1}]$, $S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$, but we need to solve now for $4n$ constants. We require that $S(x_j) = f(x_j)$ and that $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$. We also require that $S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$ and $S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$, but those last two equations are only valid for $j = 0, 1, \ldots, n-1$. Therefore we need two more equations, and we have two options for how to get those:

- **Natural/open/free boundary conditions:** $S''(x_0) = 0 = S''(x_n)$.
- **Clamped/closed boundary conditions:** $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$.

15 Lecture 15

Nothing too interesting from this lecture.

16 Lecture 16

For either natural or clamped boundary conditions, spline interpolation exists and is unique. To find it, we can put all the coeffiecients in a matrix and solve. We are guaranteed to have a solution, because a "strictly diagonally dominant" square matrix is invertible.