module 06
curve fitting, interpolation, and polynomial approximation

ahmad f. taha

ce 2989 — numerical methods in civil & env. engineering

email: ahmad.taha@vanderbilt.edu

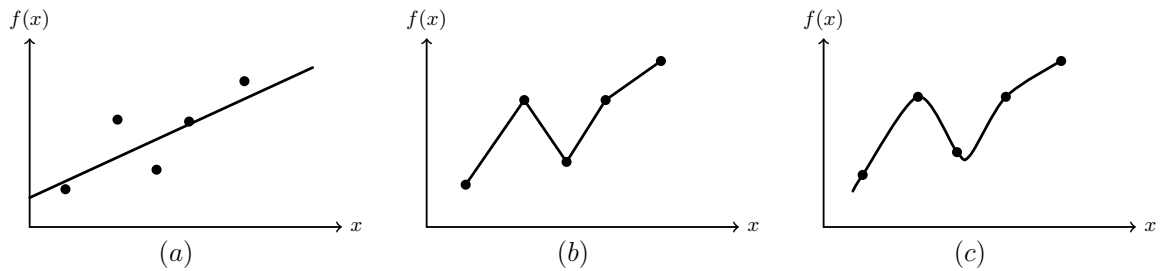
webpage: http://lab.vanderbilt.edu/taha

VANDERBILT UNIVERSITY

March 18, 2026

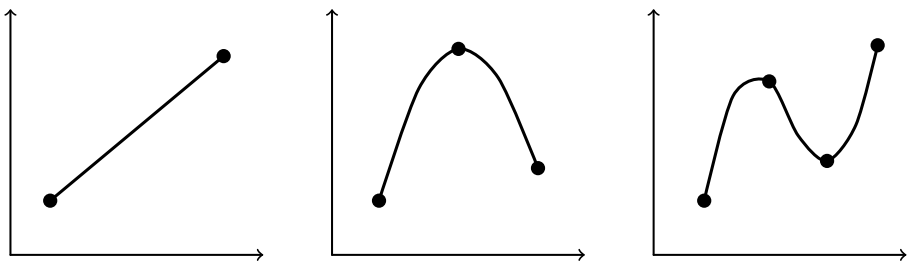
module 06 overview

- this module covers methods to fit curves (curve fitting) to discrete data
- this allows obtaining intermediate estimates
- there are two general approaches to curve fitting:
- approach 1: data exhibits a significant degree of scatter:
  - to derive a single curve that represents the general trend of data
- approach 2: data is very precise
  - to pass a curve or a series of curves through each of the points
- in engineering, two types of applications are encountered:
  - *trend analysis:* predicting values of variables—may include extrapolation beyond data points or interpolation between data points
  - *hypothesis testing:* comparing existing mathematical model with measured data
- **motivating example:** three different approaches to find a *good curve* for these five data points
- (a) least-squares regression. (b) linear interpolation. (c) curvilinear interpolation.



module 06 outline

- data usually comes in discrete form
- sometimes you wanna infer what happened between two data points
- curve fitting: obtaining estimates when you don't have data points
- **interpolation**: *the insertion of something of a different nature into something else*
- interpolation vs curve fitting—subtle differences:
  - interpolation: precisely connects all data points with a curve
  - curve fitting: finds a curve that best represents the overall trend of the data
- this module covers different techniques for interpolation/curve fitting
  - poly interpolation (newton, lagrange, error bounds)
  - divided differences and data approximation
  - splines (linear, quadratic, cubic)
  - least squares and linear/nonlinear regression
- heavily used to compute integrals/derivatives and solve ODEs



(a) 1^{st}-order (linear) connecting 2 points.

(b) 2^{nd}-order (quadratic/parabolic) connecting 3 points.

(c) 3^{rd}-order (cubic) connecting 4 points.

regression and least squares

- let's start with n data points (x_i, y_i) for $i = 1, \ldots, n$
- **objective:** predict the value y_{n+1} when new input x_{n+1} is given
- problem is called interpolation if new data point x_{n+1} is within the x_i values
- or extrapolation if it's outside the interval of x_i's
- both yield a mapping or polynomial to describe the relationship between x and y as $y = f(x)$
- in this context, no mathematical difference between extra/interpolation
- the first type of curve-fitting we'll learn isn't really data fitting
- because occasionally you wanna fit a straight line to data that isn't forming a straight line
- to determine a polynomial of degree n, you need to find the $n + 1$ coefficients of polynomial and then solve a linear system of equations $Az = b$
- where z is a vector that encodes the coefficients of the poly
- what if there's no line that fits all points? least-squares!
- notion of error distance and optimization
- we did this before so we're not gonna spend much time here

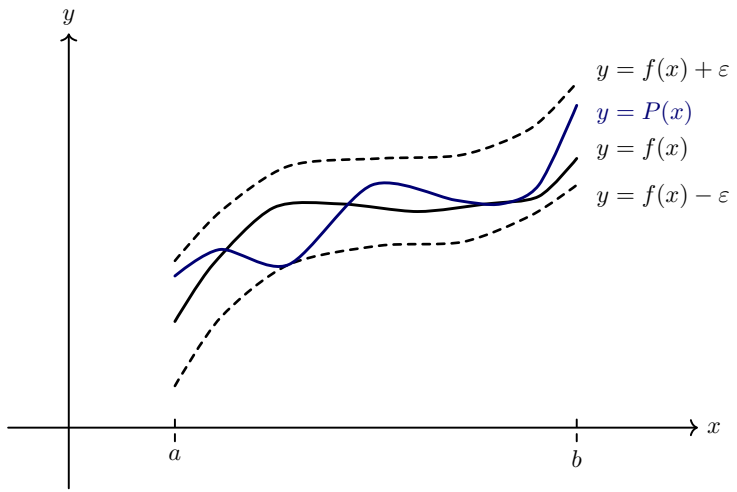
basics and intro

- problem to be studied: approximating functions via a polynomial
- why do we need that?
- polys are nice, cute, easy to derive, integrate, manipulate, factor, etc...
- **theoretical result 1:** a continuous function can be approximated by a poly
- **theoretical result 2:** polynomials of degree n interpolating values at $n+1$ distinct data points are all the same

weierstrass approximation theorem

assuming that $f(x)$ is a continuous function for $x \in [a, b]$ then for each $\epsilon > 0$, there exists a polynomial $p(x)$ such that

$$|f(x) - p(x)| < \epsilon, \ \forall x \in [a, b]$$



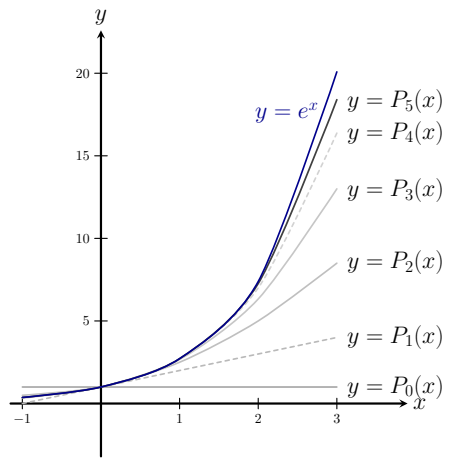
approximating polynomials via uncle taylor (no bueno)

- typical polynomial approximation: taylor series:

$$p_n(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

- example: to approximate $f(x) = e^x$ at $x_0 = 0$, we obtain these different polys for different n:

$$P_0(x) = 1,$$
$$P_1(x) = 1 + x,$$
$$P_2(x) = 1 + x + \frac{x^2}{2},$$
$$P_3(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6},$$
$$P_4(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24},$$
$$P_5(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120},$$
$$\cdots$$



- observation: even for higher-degree polys, error becomes too large away from the operating point
- so taylor approximations are good polys only around x_0
- but a good approximation/interpolation is good in an interval, not just x_0

more theoretical results

jackson's inequality

if function $f(x)$ is differentiable with a continuous kth derivative $f^{(k)}$, then we can obtain the following error bound

$$||f - p_n|| \leq \frac{\pi}{2} \frac{1}{(n+1)^k} ||f^{(k)}||$$

where n is the order of the approximating polynomial

uniqueness of polys

Let $x_0, x_1, \ldots, x_n$ be distinct real numbers, then for arbitrary values $y_0, y_1, \ldots, y_n$ there is a unique polynomial p_n of degree at most n such that $p_n(x_i) = y_i$ for $i = 0, 1, \ldots, n$.

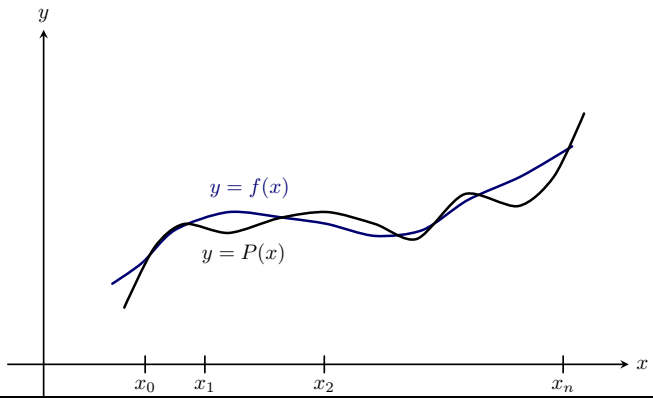
- in simpler words, if you have $n+1$ input-output data points, there will always be a **unique** polynomial
- and this polynomial can be given via the lagrange and newton polynomials
- interpolation has soooooo many applications in: engineering, comp sci, filmography, graphics, encryption, etc..

linear lagrange polys

- we will learn the first type of approximating polys: linear lagrange
- assume you know two data points for function $y = f(x)$: (x_0, y_0) and (x_1, y_1)
- defining $L_0(x) = \dfrac{x - x_1}{x_0 - x_1}, \quad L_1(x) = \dfrac{x - x_0}{x_1 - x_0}$
- notice how $L_0(x_0) = 1, L_0(x_1) = 0, L_1(x_0) = 0, L_1(x_1) = 1$
- then by definition we can write linear lagrange polys (llp) of $y = f(x)$ as

$$p(x) = y_0 L_0(x) + y_1 L_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}$$

- what do you notice?
- **example:** obtain an llp that passes through $(2, 4)$ and $(5, 1)$ (solution is $p(x) = -x + 6$)
- lp: unique polynomial of lowest degree that interpolates a given data set
- how do we construct an nth order lagrange poly?

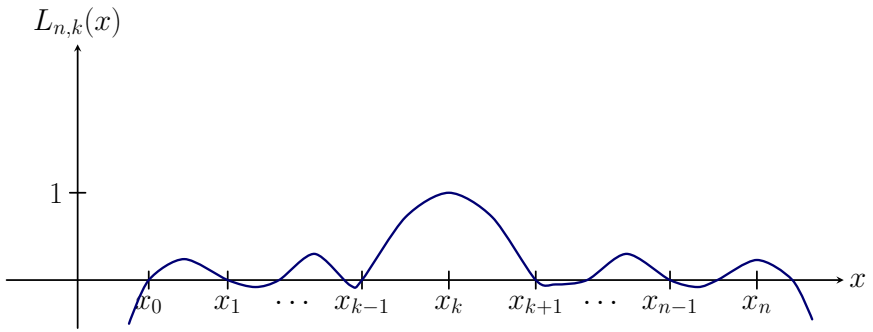


constructing nth order lagrange polys

- consider we have $n+1$ $(x_i, f(x_i))$ data points (remember the theorem)
- these data points are all different, i.e., $x_i \neq x_j$
- we need to find the *basis* or *coefficients of the lagrange polys*
- first, define the basis as for each $k = 0, 1, \ldots, n$:

$$L_{n,k}(x) = \frac{(x - x_0)}{(x_k - x_0)} \cdots \frac{(x - x_{k-1})}{(x_k - x_{k-1})} \frac{(x - x_{k+1})}{(x_k - x_{k+1})} \cdots \frac{(x - x_n)}{(x_k - x_n)} = \prod_{\substack{0 \leq i \leq n \\ i \neq k}} \frac{x - x_i}{x_k - x_i}$$

- notice how the basis $L_{n,k}(x_j) = 1$ if $j = k$ and $L_{n,k}(x_j) = 0$ if $j \neq k$



unique lagrange polys

consider distinct $n+1$ $(x_i, f(x_i))$ points, then a unique poly $p(x)$ of degree at most n exists with $f(x_k) = p_n(x_k)$ for each $k = 0, 1, \ldots, n$ and this poly is:

$$p_n(x) = f(x_0)L_{n,0}(x) + f(x_1)L_{n,1}(x) + \ldots + f(x_n)L_{n,n}(x)$$

example

- given data points $x_0 = 2, x_1 = 2.75, x_2 = 4$, find the second order lagrange poly for $f(x) = x^{-1}$ and then approximate $f(3)$
- first, need to find the basis/coefficients:

$$L_0(x) = \prod_{\substack{0 \le i \le 2 \\ i \neq 0}} \frac{x - x_i}{x_0 - x_i}, \quad L_1(x) = \prod_{\substack{0 \le i \le 2 \\ i \neq 1}} \frac{x - x_i}{x_1 - x_i}, \quad L_2(x) = \prod_{\substack{0 \le i \le 2 \\ i \neq 2}} \frac{x - x_i}{x_2 - x_i}$$

$$L_0(x) = \frac{(x - 2.75)(x - 4)}{(2 - 2.75)(2 - 4)} = \frac{2}{3}(x - 2.75)(x - 4), \; L_1(x) = \frac{(x - 2)(x - 4)}{(2.75 - 2)(2.75 - 4)} = \ldots$$

$$L_2(x) = \frac{2}{5}(x - 2)(x - 2.75)$$

- we can now write:

$$p_2(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) = \frac{1}{22}x^2 - \frac{35}{88}x + \frac{49}{44}$$

- then $f(3) \approx P_2(3) = 29/88 \approx 0.329$ and $|f(3) - p_2(3)| \approx 0.034$
- **exercise**: find the poly fit using $Ax = b$ starting with $p_2(x) = ax^2 + bx + c$

another example

- what happens if you wanna obtain the nth order lagrange poly for an nth order polynomial?
- **example:** interpolate $f(x) = x^2$ for $x = 1, 2, 3$
- can computed as

$$L_0(x) = \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} = \tfrac{1}{2}x^2 - \tfrac{5}{2}x + 3,$$
$$L_1(x) = \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} = -x^2 + 4x - 3,$$
$$L_2(x) = \frac{x-1}{3-1} \cdot \frac{x-2}{3-2} = \tfrac{1}{2}x^2 - \tfrac{3}{2}x + 1$$

and hence

$$p_2(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 4 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 9 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2} = x^2 !!!!!!!$$

- we recovered the same function!! similar to taylor series approximation!
- (this exercise is practically worthless because we never approximate polynomials with identical-degree polynomial)

lagrange poly: error analysis

lagrange poly truncation error

the relationship between the function and its nth order lagrange poly approximation can be written as

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

where $x_0, x_1, \dots, x_n$, and ξ are all in interval $[a, b]$

- what do you notice?
- how does it compare to taylor series truncation error?
- well recall that for taylor series approximation, the remainder has the form

$$\left| \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1} \right|$$

- which method produces bigger errors?
- error in taylor approx. decreases as higher derivatives are considered
- for lagrange polys, error depends on the distribution of data points
- remember: lagrange produces global approx, whereas taylor gives local

example on error analysis

- **example:** compute the error with the interpolation of $f(x) = x^{-1}$ with $x_i = 2, 2.75, 4$ and consider that $[a.b] = [2, 4]$
- via the above theorem

$$|f(x) - p_n(x)| = \left| \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_{n-1}) \right| = \left| \xi^{-4} \underbrace{(x-2)(x-2.75)(x-4)}_{h(x)} \right|$$

- maximum value of ξ^{-4} is $\frac{1}{16}$
- how to determine the max of $|h(x)|$? high-school calculus!!
- compute derivative, find critical points, etc..
- you will find that max $|h(x)|$ happens at the critical point $x^* = 3.5$ and $h(3.5) = -9/16$ then

$$|f(x) - p_n(x)| \leq \frac{1}{16}\frac{9}{16} = \frac{9}{256} \approx 0.0351$$

- consistent with the previous example $|f(3) - p_2(3)| \approx 0.034 < 0.0351$

poly interpolation: divided differences

- consider that you have already constructed the nth order lagrange poly given $n+1$ data points
- what happens if you now have an additional data point?
- unfortunately, you need to recompute **all** lagrange poly coefficients again
- is there a more efficient way for accommodating incoming new data?
- the nth order interpolation $p_n(x)$ can be written as

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

- how to compute coefficients a_k? notice: for $p_n(x_0) = f(x_0)$ so $\boxed{a_0 = f(x_0)}$
- and $p_n(x_1) = f(x_1) \Rightarrow a_1(x_1 - x_0) = f(x_1) - f(x_0)$ then

$$\boxed{a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}}$$

- what's a_2?

$$\boxed{a_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}}$$

- so basically you can derive divided difference points from lagrange polys

newton polys + divided differences

- **definitions:** divided differences of function $f(x)$ are defined as
  1. zeroth divided difference of $f(x)$ at x_i:
  $$f[x_i] = f(x_i)$$
  2. first divided difference of $f(x)$ at x_i and x_{i+1}:
  $$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$
  3. second divided difference of $f(x)$ at x_i, x_{i+1}, x_{i+2}:
  $$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$
  4. kth divided difference:
  $$f[x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$
- you can now see that $a_0 = f[x_0]$, $a_1 = f[x_0, x_1]$, or more generally: $a_k = f[x_0, x_1, \ldots, x_k]$
- **outcome:** the lagrange polys $p_n(x)$ for data points $x_{0,1,\ldots,n}$ can be written as

$$\boxed{p_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0)(x - x_1)\ldots(x - x_{k-1})}$$

divided differences table

$$p_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0)(x - x_1)\ldots(x - x_{k-1})$$

x	f[x]	First divided differences	Second divided differences	Third divided differences
x_0	$f[x_0]$			
x_1	$f[x_1]$	$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$		
x_2	$f[x_2]$	$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$	$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$	
x_3	$f[x_3]$	$f[x_2, x_3] = \frac{f[x_3] - f[x_2]}{x_3 - x_2}$	$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1}$	$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0}$
x_4	$f[x_4]$	$f[x_3, x_4] = \frac{f[x_4] - f[x_3]}{x_4 - x_3}$	$f[x_2, x_3, x_4] = \frac{f[x_3, x_4] - f[x_2, x_3]}{x_4 - x_2}$	$f[x_1, x_2, x_3, x_4] = \frac{f[x_2, x_3, x_4] - f[x_1, x_2, x_3]}{x_4 - x_1}$
x_5	$f[x_5]$	$f[x_4, x_5] = \frac{f[x_5] - f[x_4]}{x_5 - x_4}$	$f[x_3, x_4, x_5] = \frac{f[x_4, x_5] - f[x_3, x_4]}{x_5 - x_3}$	$f[x_2, x_3, x_4, x_5] = \frac{f[x_3, x_4, x_5] - f[x_2, x_3, x_4]}{x_5 - x_2}$
x_6	$f[x_6]$	$f[x_5, x_6] = \frac{f[x_6] - f[x_5]}{x_6 - x_5}$	$f[x_4, x_5, x_6] = \frac{f[x_5, x_6] - f[x_4, x_5]}{x_6 - x_4}$	$f[x_3, x_4, x_5, x_6] = \frac{f[x_4, x_5, x_6] - f[x_3, x_4, x_5]}{x_6 - x_3}$

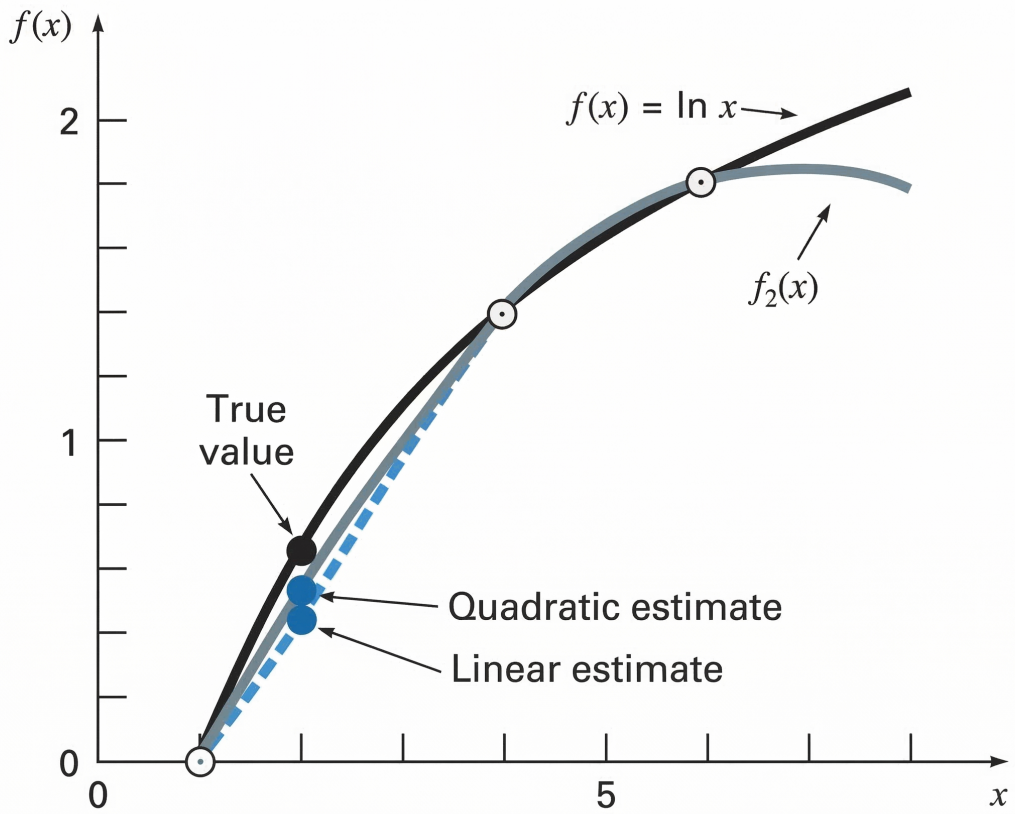
- **example:** construct an interpolation for a function with these data points: $(3, 1), (1, -3), (5, 2), (6, 4)$
- to automate this, construct the table (called ***newton tableau of div. diff.***):

$$p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) = 1 + 2(x - 3) - \tfrac{3}{8}(x - 3)(x - 1) + \tfrac{7}{40}(x - 3)(x - 1)(x - 5) = \ldots$$

i	x_i	$f(x_i)$	$f[x_{i-1}, x_i]$	$f[x_{i-2}, x_{i-1}, x_i]$	$f[x_{i-3}, x_{i-2}, x_{i-1}, x_i]$
0	3	1			
1	1	-3	2		
2	5	2	5/4	$-3/8$	
3	6	4	2	3/20	7/40

example

- comparing linear and quadratic interpolation of $y = f(x) = \ln(x)$ from $x = 1$ to $x = 4$
- the boundaries or used data points make a big impact on how accurate the interpolation is
- comparing the two, you get:



comments, discussions

divided difference poly error

similar to lagrange polys, the relationship between the function and its nth order divided difference approximation is

$$f(x) = p_n(x) + \underbrace{\frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n)}_{R_n}$$

where $x_0, x_1, \dots, x_n$, and ξ are all in interval $[a, b]$

- but this requires knowing $f(x)$ AND that $f(x)$ is differentiable $n+1$ times!
- one can prove that $R_n = f[x, x_n, x_{n-1}, \dots, x_0](x - x_0)(x - x_1)\dots(x - x_n)$
- but this is not useful because it's a function of x
- solution: with an additional point x_{n+1} we can estimate error R_n:

$$R_n \approx f[x_{n+1}, x_n, x_{n-1}, \dots, x_0](x - x_0)(x - x_1)\dots(x - x_n)$$

- basically divided differences allow you to pick the order of the polynomial given $n+1$ points—can pick a linear, quadratic, cubic polys
- more data comes in? awesome!

more discussions

- if the function $f(x)$ is of degree k, then $f[x_0, x_1, \ldots, x_n] = 0$ for all $n > k$
- why?? well, look at the divided difference table and assume $f(x)$ is linear
- in general, we can relate the nth divided difference with the nth derivative:

$$f[x_n, x_{n-1}, \ldots, x_0] = f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}$$

  where $x_i, \xi \in [a, b]$
- btw, why couldn't we apply least squares to obtain $p_n(x) = a_0 + a_1 x + \ldots a_n x^n$?
- the least squares problem can be written as $Az = b$ where z is the vector of the coefficients a_i...basic curve fitting item well, fundamentally, least squares interpolation/fitting versus lagrange/newton do completely different things
  - least squares yield polys that do not necessarily pass through all data points
  - least squares only useful when dealing with noisy data (good and bad)
  - least squares can be more accurate, btw
  - least squares requires solving $Ax = b$ hence more computational time than lagrange/divided diff
  - least squares is actually very ill-conditioned
  - divided difference can automatically accommodate new data points to refine the approximation; least squares can't

more discussions x 2

- so we learned lagrange versus newton (divided difference) interpolation
- again, as we said before, newton's polys can be updated as new data points arrive
- so they are more accommodating
- lagrange polys are directly computed via a specific equation
- whereas divided diffs are recursive equations that build the polynomial term by term
- can derive detailed result when the data-points are equally spaced
- divided diffs are more efficient to compute, less prone to errors and rounding off issues
- *lagrangian interpolation is praised for analytic utility and beauty but deplored for numerical practice*[1]
- but this is not actually true, because there are other forms to lagrange polys
- yes yes why are you so surprised??

[1] F. S. Acton, *Numerical methods that usually work*. American Mathematical Society, July 31 2020.

academics and shit-posting

don't be fooled into thinking that academics don't enjoy some shit-talking:[2]

Barycentric Lagrange Interpolation*

Jean-Paul Berrut†
Lloyd N. Trefethen‡

Dedicated to the memory of Peter Henrici (1923–1987)

Abstract. Barycentric interpolation is a variant of Lagrange polynomial interpolation that is fast and stable. It deserves to be known as the standard method of polynomial interpolation.

Key words. barycentric formula, interpolation

AMS subject classifications. 65D05, 65D25

DOI. 10.1137/S0036144502417715

1. Introduction. "Lagrangian interpolation is praised for analytic utility and beauty but deplored for numerical practice." This heading, from the extended table of contents of one of the most enjoyable textbooks of numerical analysis [1], expresses a widespread view.

In the present work we shall show that, on the contrary, the Lagrange approach is in most cases the method of choice for dealing with polynomial interpolants. The key is that the Lagrange polynomial must be manipulated through the formulas of *barycentric interpolation*. Barycentric interpolation is not new, but most students, most mathematical scientists, and even many numerical analysts do not know about it. This simple and powerful idea deserves a place at the heart of introductory courses and textbooks in numerical analysis.[1]

[2] Berrut, Jean-Paul, and Lloyd N. Trefethen. *Barycentric lagrange interpolation*. SIAM review 46.3 (2004): 501-517. https://epubs.siam.org/doi/pdf/10.1137/S0036144502417715

hermite interpolation: derivatives matter

- in the previous methods, we only cared about *matching* x_i and $f(x_i)$
- but in many applications, derivatives/rates/accelerations are more important than matching the data
- that is we want $p_n'(x)$ to match $f'(x)$
- **problem statement**: find $p_n(x)$ such that for $n+1$ data points $(x_i, f(x_i))$, we have

$$\text{HERMITE-INTERPOLATION:} \quad p_n(x_i) = f(x_i), \quad p_n'(x_i) = f'(x_i), \quad i = 0, 1, \ldots, n$$

- **example:** say you have two data points x_0 and x_1, then now you have to satisfy four conditions:

$$p_n(x_0) = f(x_0), p_n(x_1) = f(x_1), \quad p_n'(x_0) = f'(x_0), p_n'(x_1) = f'(x_1)$$

- so we have four equations...so what order should $p(x)$ be?
- desired polynomial is in degree 3, since it's parameterized by four parameters
- what if we have $n+1$ data points as before? how many conditions do we obtain? $2n+2$ conditions

hermite interpolation (cont'd)

- one possible interpolation:

$$p_3(x) = a + b(x - x_0) + c(x - x_0)^2 + d(x - x_0)^2(x - x_1)$$

  and hence $p_3'(x) = b + 2c(x - x_0) + 2d(x - x_0)(x - x_1) + d(x - x_0)^2$
- why this polynomial? why not $p_3(x) = a + bx + cx^2 + dx^3$?
- because this makes finding a, b, c, d so much easier:
- so given the four equalities/conditions, we obtain ($h = x_1 - x_0$):

$$f(x_0) = a,\ f'(x_0) = b,\ f(x_1) = a + bh + ch^2,\ f'(x_1) = b + 2ch + dh^2$$

hermite interpolation theorem

given $n + 1$ distinct data points $(x_i, f(x_i))$, the hermite poly of degree $2n + 1$ is

$$p_{2n+1}(x) = \sum_{i=0}^{n} f(x_i)p_{n,i}(x) + \sum_{i=0}^{n} f'(x_i)\hat{p}_{n,i}(x)$$

$$p_{n,i}(x) = \left(1 - 2(x - x_i)L'_{n,i}(x_i)\right) L^2_{n,i}(x),\quad \hat{p}_{n,i}(x) = (x - x_i)L^2_{n,i}(x)$$

where $L_{n,i}(x)$ is the ith lagrange poly of degree n

- **home exercise:** to see how this works, show that $p_{2n+1}(x)$ actually satisfies the conditions given the data points

hermite poly: an example

- obtain the hermite poly for the following data points:

$$\{x_i, f(x_i), f'(x_i)\} = \{(\underbrace{1}_{x_0}, \overbrace{1}^{f(x_0)}, \underbrace{-1}_{f'(x_0)}), (\underbrace{2}_{x_1}, \overbrace{-3}^{f(x_1)}, \underbrace{2}_{f'(x_1)})\}$$

- in this case, $n = 1$ and hence we have

$$p_3(x) = \sum_{i=0}^{1} f(x_i)p_{1,i}(x) + \sum_{i=0}^{1} f(x_i)\hat{p}_{1,i}(x) = f(x_0)p_{1,0} + f(x_1)p_{1,1} + f'(x_0)\hat{p}_{1,0} + f'(x_1)\hat{p}_{1,1}$$

- need to compute

$$p_{1,0}(x) = \big(1 - 2(x - x_0)L'_{1,0}(x)\big)\, L^2_{1,0}(x), \quad \hat{p}_{1,0}(x) = (x - x_0)L^2_{1,0}(x)$$

$$p_{1,1}(x) = \big(1 - 2(x - x_1)L'_{1,1}(x)\big)\, L^2_{1,1}(x), \quad \hat{p}_{1,1}(x) = (x - x_1)L^2_{1,1}(x)$$

- remember that $L_{1,0}(x) = -(x - 2),\ L_{1,1}(x) = x - 1$ then we have:

$$p_{1,0}(x) = (1 - 2(x-1)(-1))(x-2)^2 = (2x-1)(x-2)^2, p_{1,1}(x) = -(2x-5)(x-1)^2$$

$$\hat{p}_{1,0}(x) = (x-1)(x-2)^2, \quad \hat{p}_{1,1}(x) = (x-2)(x-1)^2$$

- hence

$$p_3(x) = p_{1,0} - 3p_{1,1} - 1\hat{p}_{1,0} + 2\hat{p}_{1,1} = 9x^3 - 39x^2 + 50x - 19$$

hermite divided diff table

- you can also obtain the hermite poly via the divided difference table
- way less tedious than writing the equations down
- since each data point x_i corresponds with two values $f(x_i)$ and $f'(x_i)$, we denote $z_{2i} = z_{2i+1} = x_i$ for $i = 0, 1, \ldots, n$
- we can construct the divided difference table using $z_0, z_1, z_2, \ldots, z_{2n+1}$
- note that for $z_{2i} = z_{2i+1} = x_i$, div. diff. $f[z_{2i}, z_{2i+1}] = \frac{f[z_{2i+1}] - f[z_{2i}]}{z_{2i+1} - z_{2i}}$ is undefined so we replace it with $f[z_{2i}, z_{2i+1}] = f'(x_i)$
- and the hermite divided diff poly can be written as:

$$p_{2n+1}(x) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, z_1, \ldots, z_k](x - z_0)(x - z_1)\ldots(x - z_{k-1})$$

z	$f[z]$	First divided differences	Second divided differences
$z_0 = x_0$	$f[z_0] = f(x_0)$		
$z_1 = x_0$	$f[z_1] = f(x_0)$	$f[z_0, z_1] = f'(x_0)$	
$z_2 = x_1$	$f[z_2] = f(x_1)$	$f[z_1, z_2] = \frac{f[z_2] - f[z_1]}{z_2 - z_1}$	$f[z_0, z_1, z_2] = \frac{f[z_1, z_2] - f[z_0, z_1]}{z_2 - z_0}$
$z_3 = x_1$	$f[z_3] = f(x_1)$	$f[z_2, z_3] = f'(x_1)$	$f[z_1, z_2, z_3] = \frac{f[z_2, z_3] - f[z_1, z_2]}{z_3 - z_1}$
$z_4 = x_2$	$f[z_4] = f(x_2)$	$f[z_3, z_4] = \frac{f[z_4] - f[z_3]}{z_4 - z_3}$	$f[z_2, z_3, z_4] = \frac{f[z_3, z_4] - f[z_2, z_3]}{z_4 - z_2}$
$z_5 = x_2$	$f[z_5] = f(x_2)$	$f[z_4, z_5] = f'(x_2)$	$f[z_3, z_4, z_5] = \frac{f[z_4, z_5] - f[z_3, z_4]}{z_5 - z_3}$

example + error bound

- obtain the hermite divided difference poly for the following data points:

$$\{x_i, f(x_i), f'(x_i)\} = \{(1, 1, -1), (2, -3, 2)\}$$

- result should agree with the lagrange-based hermite poly
- **solution:**

$$\begin{aligned} z0 = 1,\ f[z_0] = 1\ f[z_0, z_1] = -1,\ f[z_0, z_1, z_2] = -3,\ f[z_0, z_1, z_2, z_3] = 9 \\ z_1 = 1,\ f[z_1] = 1,\ f[z_1, z_2] = -4,\ f[z_1, z_2, z_3] = 6 \\ z_2 = 2,\ f[z_2] = -3,\ f[z_2, z_3] = 2 \\ z_3 = 2,\ f[z_3] = -3 \end{aligned}$$

- hence

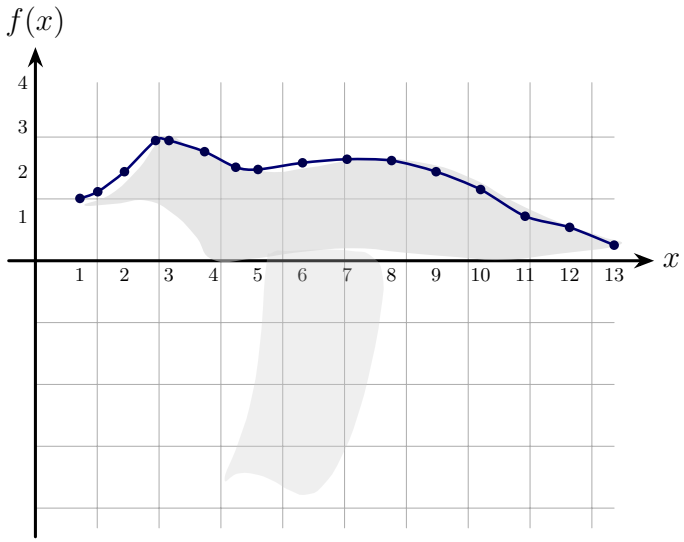
$$p_3(x) = 1 - (x-1) - 3(x-1)^2 + 9(x-1)^2(x-2) = 9x^3 - 39x^2 + 50x - 19$$

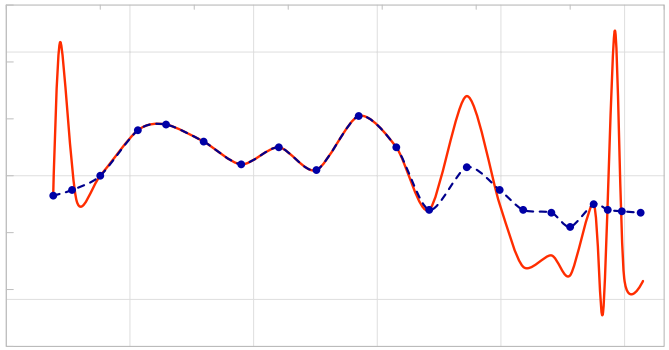
- **error bounds:** the hermite polys incur error

$$\boxed{R_n = f(x) - p_{2n+1}(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \prod_{i=0}^{n} (x - x_i)^2}$$

splines

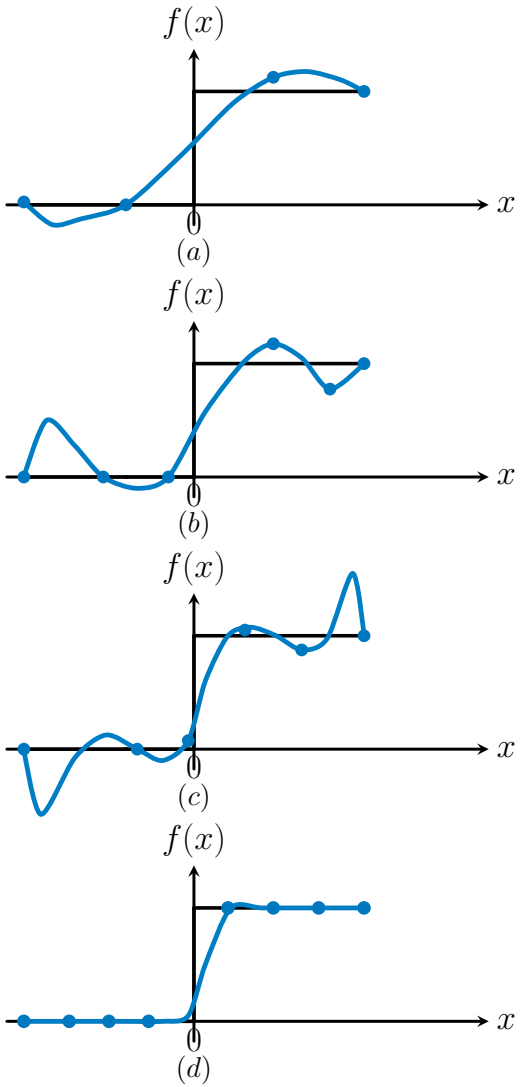
- in the previous discussions, we approximated $n+1$ data points with a single poly of degree n
- in reality, overshoot (large oscillations) + round off errors can happen
- resulting in large errors for the single poly of degree n
- **solution:** obtain lower-order polys for subsets of data points
- **example:** third-order curves are called cubic splines
- need the *connections* between splines to be *smooth*
- duck example: a 20th order lagrange polynomial isn't good enough





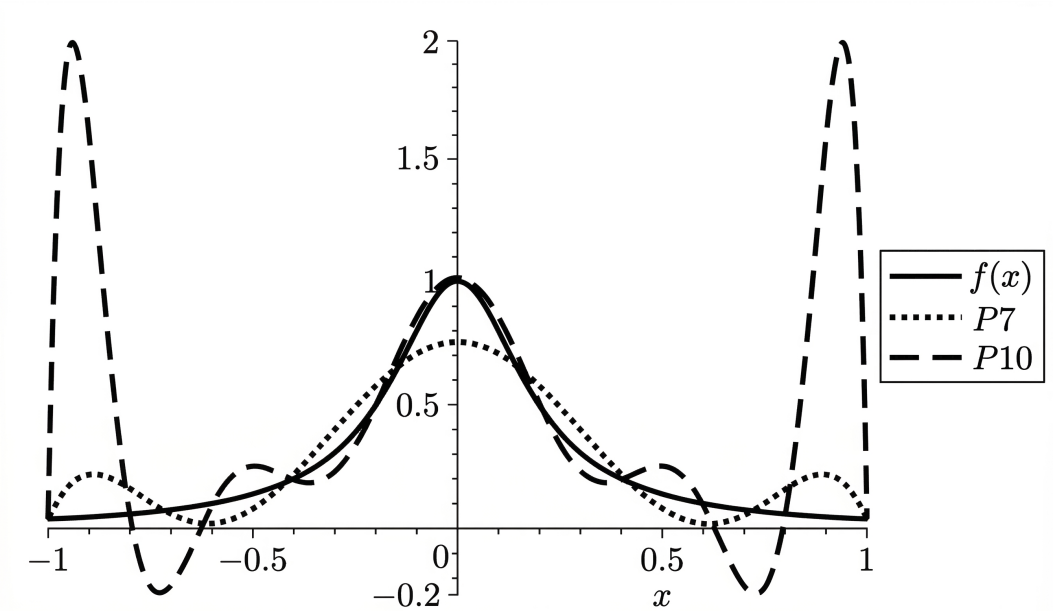
example

- where we show that two connected, simple linear interpolation performs better than higher order polynomials:



deriving splines

- we could derive first-order (piece-wise linear) splines, quadratic, cubic splines, etc..
- basically time-varying lower order polys approximating subsets of data points
- so the **objectives** are that the splines be:
  - piecewise lower-order polys
  - continuously differentiable over domain to ensure smoothness
  - no derivative information of the original function are required
- one more example before we talk about the methods
- for function $= (1 + 25x^2)^{-1}$, $x \in [-1, 1]$ and points x_i sampled uniformly, lagrange/divided difference approximations diverge:



linear splines

- **defining partitions:** a partition of interval $[a, b]$ is an ordered sequence $\{x_i\}$ for $i = 0, 1, \ldots, n$ such that $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ where x_i are all known and called *nodes*

defining *splines*

an approximating function $s(x)$ is a spline of degree k on $[a, b]$ if:

- domain of $s(x)$ is $[a, b]$ (domain of the spline is consistent with the data points)
- there exists partition $\{x_i\}$ for $i = 0, 1, \ldots, n$ in $[a, b]$ such that on each sub-interval $[x_{i-1}, x_i]$ we have $s(x)$ is polynomial of order k (each approximating poly is at most of degree k)
- $s(x), s'(x), \ldots, s^{(k-1)}(x)$ are all continuous on (a, b) (they're smooth)

- most widely used splines are cubic splines but let's first learn about the linear and quadratic ones

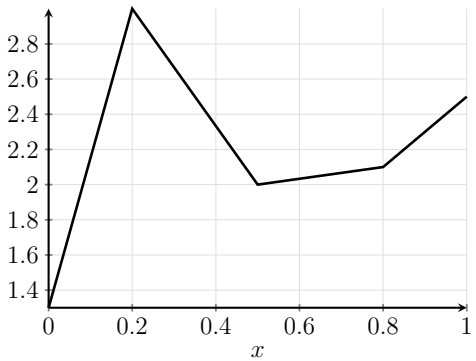
linear and quadratic splines

- **linear splines:** given $n+1$ data points (x_i, y_i), the linear splines $l(x)$ are:

$$l(x) = y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}), \quad x \in [x_{i-1}, x_i]$$

- does this linear spline satisfy the conditions? why does it make sense?
- **example:** construct linear spline for this data:

$$(x_i, y_i) = \{(0, 1.3), (0.2, 3)(0.5, 2), (0.8, 2.1), (1, 2.5)\}$$



$$f(x) = \begin{cases} 1.3 + 8.5x, & x < 0.2, \\ \frac{11}{3} - \frac{10x}{3}, & x < 0.5, \\ \frac{13}{6} + \frac{x}{3}, & x < 0.8, \\ 0.5 + 2.0x, & \text{otherwise.} \end{cases}$$

error analysis of linear splines

the error for first degree splines is given by this bound

$$|f(x) - l(x)| \leq \frac{\max_{x \in (a,b)} |f''(x)|}{8} \max_{1 \leq i \leq n} (x_i - x_{i-1})^2$$

quadratic splines

- a second order spline $q(x)$ is a piecewise quadratic function
- its derivative is continuous on interval (a, b)
- given by: $q_i(x) = a_i x^2 + b_i x + c, \quad x \in [x_{i-1}, x_i], \quad i = 1, 2, \ldots, n$
- how many parameters for $q(x)$? $3n$ is the answer! how to determine the coefficients?
- well for each i or interval, spline has to satisfy the following conditions:

$$\underbrace{q_i(x_{i-1}) = y_{i-1}, \quad q_i(x_i) = y_i}_{i=1,\ldots,n}, \quad \underbrace{q_i'(x_i) = q_{i+1}'(x_i) = \overbrace{z_i}^{\text{unknown}}}_{i=1,\ldots,n-1}$$

- this hence results in $3n$ unknowns, $3n - 1$ equations
- to make the system of equations consistent, we can add an additional constraint such as $q'(a) = f'(a)$, $q'(b) = f'(b)$, or $q''(a) = 0$
- because $q_i'(x)$ is a linear, then $q_i'(x_{i-1}) = z_{i-1}, \; q_i'(x_i) = z_i$, and:

$$q_i'(x) = z_{i-1} + \frac{z_i - z_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}), \quad x \in [x_{i-1}, x_i]$$

- integrating the last equation w.r.t x, and using $q_i(x_{i-1}) = y_{i-1}$, we obtain

$$\boxed{y_i = q_i(x) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x - x_{i-1})^2 + z_{i-1}(x - x_{i-1}) + y_{i-1}}$$

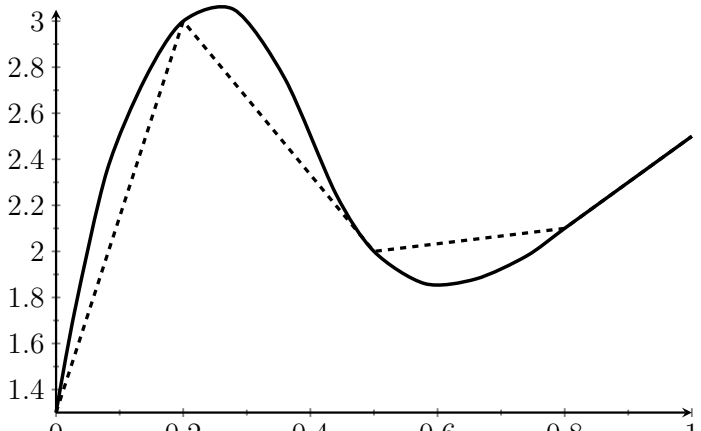
quadratic splines (cont'd)

$$y = q_i(x) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x - x_{i-1})^2 + z_{i-1}(x - x_{i-1}) + y_{i-1}$$

- need to determine z_i now so you can compute $q_i(x)$:

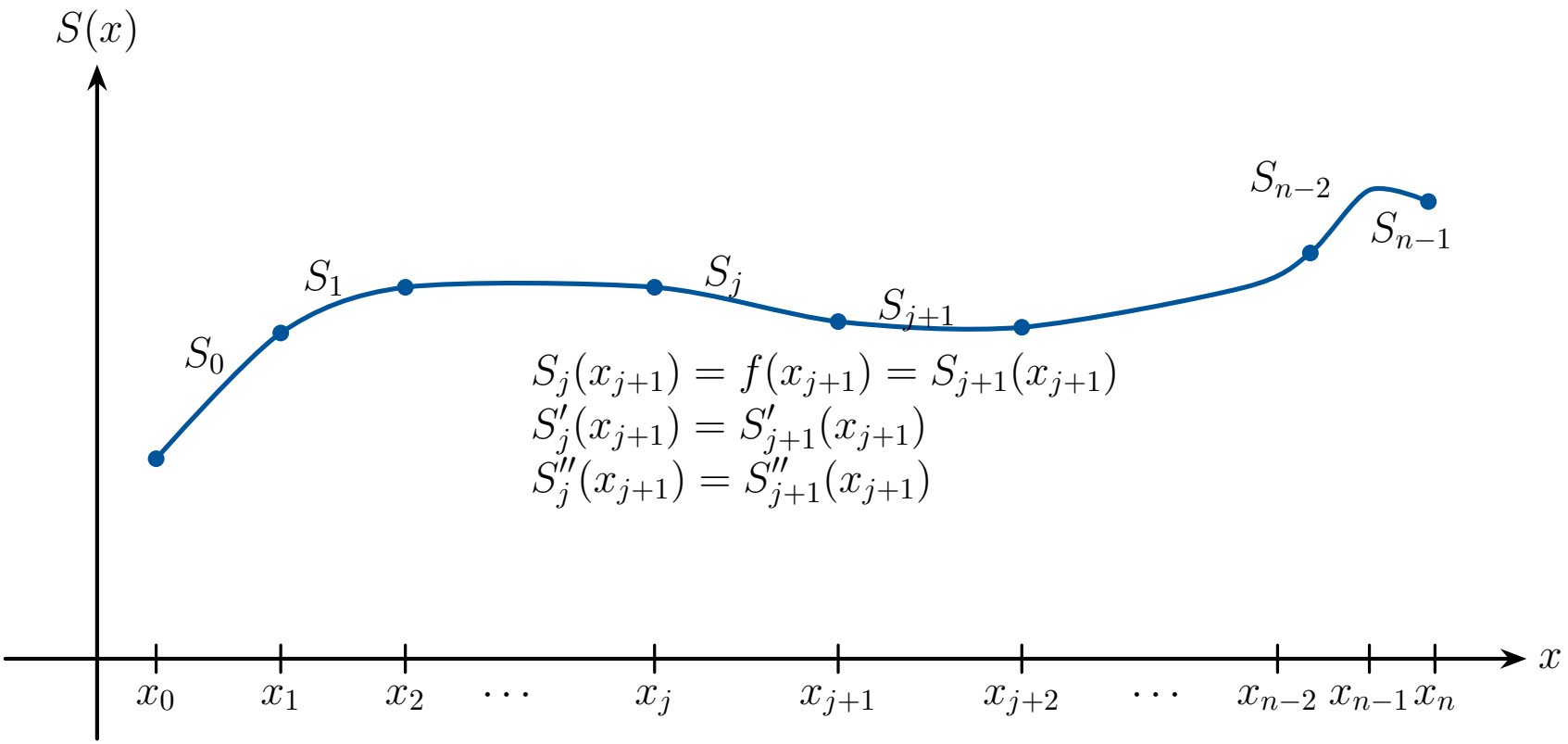
$$y_i = q_i(x_i) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x_i - x_{i-1})^2 + z_{i-1}(x_i - x_{i-1}) + y_{i-1} = (x_i - x_{i-1})\frac{z_i + z_{i-1}}{2} + y_{i-1}$$

- or $z_i = 2\dfrac{y_i - y_{i-1}}{x_i - x_{i-1}} - z_{i-1}, \quad i = 1, 2, \ldots, n$
- note that $z_0 = q_1'(x_0)$ but this value is unknown, so we need to approximate it to compute z_i and hence $q_i(x)$
- **example:** construct $q(x)$ for $\{(0, 1.3), (0.2, 3), (0.5, 2), (0.8, 2.1), (1, 2.5)\}$
- approximate $z_0 = \frac{y_1 - y_0}{x_1 - x_0} = 8.5$ then obtain $z_{1,2,3,4}$ then construct $y = q_i(x)$



cubic splines

- most common piecewise-polynomial approximation: **cubic spline**
- from the name, we note that this uses cubic interpolations that are time-varying but still continuous and second-order differentiable
- looks like this:



- why are cubic splines very popular?

constructing cubic splines

- given $n+1$ data points (x_i, y_i) $(i = 0, \ldots, n)$, the objective is to construct $c_j(x)$ (cubic spline) in every interval $[x_j, x_{j+1}]$ for $j = 0, \ldots, n-1$
- the following conditions hold
  1. $c_j(x_j) = f(x_j),\ c_j(x_{j+1}) = f(x_{j+1})$ for $j = 0, \ldots, n-1$
  2. $c'_{j+1}(x_{j+1}) = c'_j(x_{j+1})$ for $j = 0, \ldots, n-2$
  3. $c''_{j+1}(x_{j+1}) = c''_j(x_{j+1})$ for $j = 0, \ldots, n-2$
  4. boundary conditions: $c''(x_0) = c''(x_n) = 0$
- the last boundary conditions ensures the *spline is natural*
- this means that the function becomes a straight line at the end-points
- how many conditions in terms of n? total is $2n + 2(n-1) + 2 = 4n$ for $4n$ coefficients of the n cubic polynomials with 4 coefficients each
- can you derive the individual cubic polys $c_j(x) = a_j x^3 + b_j x^2 + c_j x^+ d_j$ for all intervals?
- to work with conditions (3) and (4) above, you start with the knowledge that $c''(x)$ is a straight line then integrate twice to get $c(x)$
- how to compute these parameters?

cubic splines computation

- to obtain $c(x)$, we use this expression for $h_i = x_i - x_{i-1}$ for $i = 1, \ldots, n$

$$c_i(x) = \frac{z_{i-1}(x_i - x)^3}{6h_i} + \frac{z_i(x - x_{i-1})^3}{6h_i} + \left(\frac{y_i}{h_i} - \frac{1}{6}z_i h_i\right)(x - x_{i-1}) + \left(\frac{y_{i-1}}{h_i} - \frac{1}{6}z_{i-1}h_i\right)(x_i - x)$$

- only missing thing in the equation are the z_i's which are the $c''(x)$
- with $z_0 = z_n = 0$, we can solve $Az = b$ for $z = [z_1, z_2, \ldots, z_{n-1}]$ where

$$A = \begin{bmatrix} 2(h_1 + h_2) & h_2 & & & \\ h_2 & 2(h_2 + h_3) & h_3 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ & & & h_{n-1} & 2(h_{n-1} + h_n) \end{bmatrix} \quad b = \begin{bmatrix} b_2 - b_1 \\ b_3 - b_2 \\ \vdots \\ b_n - b_{n-1} \end{bmatrix}$$

and $b_i = \frac{6}{h_i}(y_i - y_{i-1})$

- this is an easy linear system ($Az = b$) to solve for, relatively speaking
- always solvable? is A always full-rank??
- after computing the z_i's, can construct the $n - 1$ cubic polynomials $c_i(x)$

example

- the way we derived the functions relies on specific format of $c_i(x)$
- but you don't have to stick with this
- construct a *natural* cubic spline for data points: $\{(1,2),(2,3),(3,5)\}$ without following the procedure in the previous slides
- how many cubic polynomials to derive? only two since $n=2$: $c_1(x)$ for $\{(1,2),(2,3)\}$ and $c_2(x)$ for $\{(2,3),(3,5)\}$
- two cubic polys can be written as

$$c_1(x) = a_1+b_1(x-1)+c_1(x-1)^2+d_1(x-1)^3,\ c_2(x) = a_2+b_2(x-2)+c_2(x-2)^2+d_2(x-2)^3$$

- we have 8 unknowns and we can obtain 8 conditions as outlined before
- first four conditions/equations are:

$$c_1(1) = a_1 = 2,\ c_1(2) = 3 \rightarrow b_1+c_1+d_1 = 1,\ c_2(2) = a_2 = 3,\ c_2(3) = 5 \rightarrow b_2+c_2+d_2 = 2$$

- last 4 eqns. are (interior 1st/2nd derivatives are equal & *natural* spline):

$$c_1'(2) = c_2'(2) = b_1+2c_1+3d_1 = b_2,\ c_1''(2) = c_2''(2) = 2c_1+6d_1 = 2c_2,\ 0 = 2c_1,\ 0 = 2c_2+6d_2$$

- solving 8 eqns, 8 unknowns, you get:

$$c_1(x) = 2+3/4(x-1)+1/4(x-1)^3,\ \ c_2(x) = 3+3/2(x-2)+3/4(x-2)^2-1/4(x-2)^3$$

final notes

- colloquially, when folks say splines they almost always mean cubic splines
- another type of cubic splines, are called *clamped* cubic splines
- in clamped cubic splines, the derivative evaluations at the end points are given:

$$c_1'(x_0) = f'(x_0), \quad c_{n-1}'(x_n) = f'(x_n)$$

- this adds two equations but now the z_0 and z_n are not equal to zero, meaning we end up getting a system of $n+1$ equations with $n+1$ unknowns
- this depends whether the user has provided these additional data points

module summary

- we learned methods to perform interpolation given data points
- some methods require re-computation of the parameters as new data comes
- other methods can be updated continuously to produce a better approximating of black-box models of input-output mappings
- lagrange/divided differences are simple and effective
- but splines and hermites allow you to do cooler things + impose constraints on the smoothness/continuity