

## Ch 3.5: Splines

Friday, October 17, 2025 11:04 AM

In our previous discussions of interpolation, we considered a single polynomial

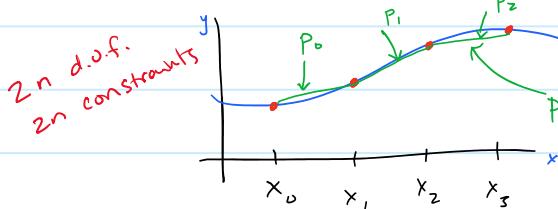
we'll always assume these are distinct

but if we have many points  $\{x_0, x_1, \dots, x_n\}$ , then we need a high-degree polynomial which can be undesirable (e.g., Runge phenomenon if equispaced nodes)

A very popular alternative is to do a piecewise polynomial interpolation.

Ex: piecewise linear

The mathematics are straightforward so we won't focus on



$P_i$ : polynomials  
 $S$  is piecewise polynomial

$$\begin{cases} P_1 & \text{if } x \in [x_0, x_1) \\ P_2 & \text{if } x \in [x_1, x_2) \\ P_3 & \text{if } x \in [x_2, x_3] \end{cases}$$

In Matlab/numpy, this is "interp1"

Downside of piecewise linear is that the approximation

is not differentiable, ie., derivative on one piece

doesn't match derivative on neighboring piece.

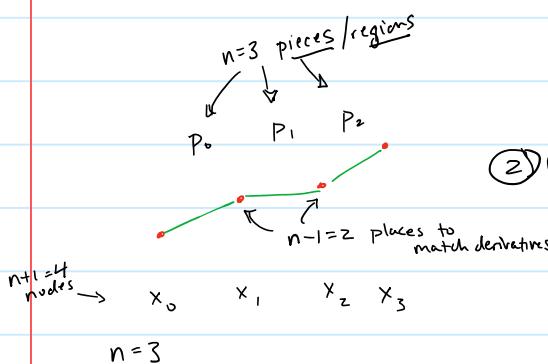
$$S(x) = \left\{ \begin{array}{ll} p_j(x) \\ \text{if } x \in [x_{j-1}, x_j] \end{array} \right\} \text{ i.e., } \textcircled{1} \text{ interpolation requirement } \quad S(x_i) = f(x_i), \quad i = 0, 1, \dots, n$$

meanings

$$\begin{aligned} p_i(x_i) &= f(x_i) \quad i = 0, 1, \dots, n-1 \\ p_i(x_{i+1}) &= f(x_{i+1}) \end{aligned} \quad \left. \right\} 2n \text{ constraints}$$

So far in piecewise linear polynomials, we have

$2n$  parameters, which matches  $2n$  constraints



### meanings

$$p_i'(x_{i+1}) = p_{i+1}'(x_{i+1}), \quad i = 0, 1, \dots, n-2 \quad \left\{ \begin{matrix} n-1 \\ \text{constraints} \end{matrix} \right.$$

Impossible w/ piecewise linear!

Going beyond linear interpolation,

What if we want to impose conditions ① and ②?

$$\text{that's } 2n + (n-1) = 3n-1 \text{ constraints}$$

Suppose we use a quadratic rather than linear polynomial on each region?

Then  $3n$  parameters but only  $3n-1$  constraints so it's not unique

unless we specify one more constraint. A boundary condition is natural,

but only have room for one ( $x_0$  or  $x_n$ , not both) ... AWKWARD!

Suppose we use a cubic polynomial? Then we have  $4n$  parameters.

We can add a lot of constraints. Let's add

(3) Have second derivatives agree on interior nodes

$$P_i'''(x_{i+1}) = P_{i+1}'''(x_{i+1}), \quad i=0, 1, \dots, n-2 \quad \left. \begin{array}{l} \\ \end{array} \right\} n-1 \text{ constraints}$$

Then (1) + (2) + (3) is  $2n + (n-1) + (n-1) = 4n-2$  constraints

So we have  $\underline{\underline{2}}$  extra degrees of freedom. This is a nice symmetric case. We can impose

(4a) "Natural" or "Free" Boundaries,

$$\begin{aligned} S''(x_0) &= 0 && \text{i.e., no curvature at end} \\ S''(x_n) &= 0 && \text{i.e., } P_0'''(x_0) = 0, P_{n-1}'''(x_n) = 0 \end{aligned}$$

all useful.  
we'll focus  
on  
4)  
"Natural"

OR

(4b) "clamped boundary" of course only works if we can compute  $f'$   
 $S'(x_0) = f'(x_0)$   
 $S'(x_n) = f'(x_n)$  i.e.,  $P_0'(x_0) = f'(x_0)$   
 $P_{n-1}'(x_n) = f'(x_n)$

OR  
(4c) "not-a-knot"

$$\begin{aligned} S''' &\text{ continuous at } x_1 \text{ and } x_{n-1} \\ \text{i.e., } P_0'''(x_1) &= P_1'''(x_1) \\ P_{n-2}'''(x_{n-1}) &= P_{n-1}'''(x_{n-1}) \end{aligned}$$

"Knot" is synonymous with "node", meaning a  $x_i$  point

OR

(4d) "periodic"  
 $S'(x_0) = S'(x_n)$   
 $S''(x_0) = S''(x_n)$



Condition (2) says  $S \in C^1([a, b])$ , i.e., derivatives match each other

Not the same as piecewise Hermite interpolation,  
which would require derivatives of  $S$  to match those of  $f$

Conditions (1) + (2) + (3) + (4a) or (4c) or (4d) do not require knowing  $f'$

## SPLINES

A piecewise function satisfying conditions like (1), (2) and (3) is called a spline.

(Note: B-Splines are a bit different, since don't interpolate, but have advantage that if you move a node, you don't have to recompute from scratch.)

Used in NURBS for CAD (computer-aided design) programs

These and Bézier curves (used in graphics, e.g. fonts, vector graphics)

are beyond the scope of this course)

vector vs. raster  
(no resolution)

✓

(JPEG)

We'll talk exclusively about CUBIC SPLINES (1) + (2) + (3) + (4)

and in particular NATURAL CUBIC SPLINES (1) + (2) + (3) + (4a)

 **interp1** (Matlab and Scipy) takes a "cubic" option

but doesn't satisfy (3).

(all boundary conditions)

To get a cubic spline, **scipy.interpolate.CubicSpline** (python)

(python)

or **interp1** w/ "spline" option { "not-a-knot" (Matlab)}

or **Spline**

or **csape**/**csapi**

(Matlab curve-fitting toolbox)

So... we have **4n** degrees of freedom, **4n** constraints

Basic technique: Fix a basis (representation) for the polynomials

in each interval, setup a  $4n \times 4n$  linear system,

and solve in  $O(n^3)$  time.

Tricks: pick a good representation for the polynomials (for Stability)

and tricks to solve system in  $O(n)$  time

1) How to represent  $p_j(x)$

ex.

$$p_j(x) = a_j + b_j x + c_j x^2 + d_j x^3 \quad j=0, 1, \dots, n-1$$

better:

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

... so clearly  $a_j = f(x_j)$

2) Setup matrix system

$$\Rightarrow \boxed{a_j = f(x_j)} \checkmark \quad (\star)$$

Recall: ① 1a)  $P_j(x_j) = f(x_j) \quad j=0, 1, \dots, n-1$   
 1b)  $P_j(x_{j+1}) = f(x_{j+1})$

$$(2) \quad P_j'(x_{j+1}) = P_{j+1}'(x_{j+1}) \quad j=0, 1, \dots, n-2$$

$$(3) \quad P_j''(x_{j+1}) = P_{j+1}''(x_{j+1}) \quad j=0, 1, \dots, n-2$$

$$(4a) \quad P_0''(x_0) = 0 \\ P_{n-1}''(x_n) = 0$$

Note  $P_j'(x) = b_j + 2c_j(x-x_j) + 3d_j(x-x_j)^2, \quad P_j''(x) = 2c_j + 6d_j(x-x_j)$

so  $\underline{P_j'(x_j) = b_j} \quad (\star\star) \quad \text{and likewise } P_j''(x_j) = 2c_j \quad (\star\star\star)$

Notation:  $\underline{h_j = \Delta x_j}, \quad \text{i.e.,} \quad \underline{h_j = x_{j+1} - x_j}$

Note 1b can now be written  $\underbrace{P_j(x_{j+1})}_{j=0, 1, \dots, n-1} = f(x_{j+1}) = \underbrace{P_{j+1}(x_{j+1})}_{a_{j+1}} \quad j=0, 1, \dots, n-1$   
 $\text{RHS } a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 = \text{LHS } a_{j+1} \quad \text{via } (\star) \quad (\text{w/ } a_n := f(x_n))$

And (2), with  $(\star\star)$ , gives

$$\underbrace{P_j'(x_{j+1})}_{j=0, 1, \dots, n-1} = \underbrace{P_{j+1}'(x_{j+1})}_{b_{j+1}} \quad \text{via } (\star\star) \quad (\text{w/ } b_n = s'(x_n))$$

And (3), with  $(\star\star\star)$ , gives

$$\underbrace{P_j''(x_{j+1})}_{j=0, 1, \dots, n-1} = \underbrace{P_{j+1}''(x_{j+1})}_{2c_{j+1}} \quad \text{via } (\star\star\star) \quad (\text{w/ } c_n = s''(x_n)_{1/2})$$

$$\text{i.e., } \textcircled{C} \quad c_j + 3d_j h_j = c_{j+1}$$

$$\Rightarrow d_j = \frac{c_{j+1} - c_j}{3h_j}$$

Plugging into

$$\left. \begin{array}{l} j=0, 1, \dots, n-1 \\ \hline \end{array} \right\}$$

and (A)  $a_{j+1} = a_j + b_j h_j + h_j^2 / 3 (2c_j + c_{j+1})$

Recall  $a_j$  are known!

(B)  $b_{j+1} = b_j + h_j (c_j + c_{j+1})$

So  $b_j = \frac{1}{h_j} (a_{j+1} - a_j) - h_j / 3 (2c_j + c_{j+1}) \quad (j=0, 1, 2, \dots, n-1)$

and  $b_{j+1} = \frac{1}{h_{j+1}} (a_{j+2} - a_{j+1}) - h_{j+1} / 3 (2c_{j+1} + c_{j+2}) \quad (j=-1, 0, 1, \dots, n-2)$  for  $j=0, \dots, n-2$

*both hold*

*you don't need to be able to recreate this on an exam*

So now (B) becomes (after grouping terms)

\*  $\underbrace{h_j c_j}_{j=0, 1, \dots, n-2} + 2(h_j + h_{j+1}) c_{j+1} + h_{j+1} c_{j+2} = \frac{3}{h_{j+1}} (a_{j+2} - a_{j+1}) - \frac{3}{h_j} (a_{j+1} - a_j)$

So... Solve these  $(n-1)$  equations for  $c$  Known!

then plug into earlier equations to find  $\underline{b}$  and  $\underline{d}$ .

Also need boundary condition 4a  $P_0''(x_0) = 0$  and  $P_{n-1}''(x_n) = 0$

$$P_i''(x) = 2c_i + 6d_i(x - x_i)$$

$$\text{so } P_0''(x_0) = 2c_0, \text{ so } P_0''(x_0) = 0 \text{ means } \underline{c_0 = 0}.$$

$$\text{also, recall we defined } C_n = S''(x_n)/2, \text{ so } S''(x_n) = 0 \Rightarrow c_n = 0.$$

Thus:

$$c_0 = 0$$

$$= y_0$$

$$h_0 c_0 + 2(h_0 + h_1) c_1 + h_1 c_2 = \frac{3}{h_1} (a_2 - a_1) - \frac{3}{h_0} (a_1 - a_0) = y_1$$

$$h_1 c_1 + 2(h_1 + h_2) c_2 + h_2 c_3 = \frac{3}{h_2} (a_3 - a_2) - \frac{3}{h_1} (a_2 - a_1) = y_2$$

:

$$h_{n-2} c_{n-2} + 2(h_{n-2} + h_{n-1}) c_{n-1} + h_n c_n = \frac{3}{h_{n-1}} (a_n - a_{n-1}) - \frac{3}{h_{n-2}} (a_{n-1} - a_{n-2}) = y_{n-1}$$

$$c_n = 0$$

$$= y_n$$

$$\left[ \begin{array}{cccccc} 1 & 0 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & & | \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & & \\ | & & & \ddots & & \\ 0 & & & & h_{n-1} & 0 \\ & & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ & & & & 0 & 0 & 1 \end{array} \right] \left[ \begin{array}{c} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{array} \right] = \left[ \begin{array}{c} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{array} \right]$$

This is a system of  $(n+1)$  unknowns and  $(n+1)$  equations

This is a system of  $(n+1)$  unknowns and  $(n+1)$  equations

(1) Fact: it's invertible

(proof: it's "diagonally dominant", cf. ch 6.6,

and by the Gershgorin Disk theorem, there are  
aka Gershgorin Circle

no zero eigenvalues ... a special case is proven in Thm. 6.21)

(2) It's a tridiagonal matrix, and can be solved efficiently

A fact \* everyone should know

Specifically, note that solving a general system of  
linear  $n$  equations and  $n$  unknowns takes  $\mathcal{O}(n^3)$  flops

$$\begin{matrix} n \\ | \\ \begin{bmatrix} & & \\ A & & \\ & & \end{bmatrix} \end{matrix} \begin{bmatrix} x \\ | \\ b \end{bmatrix}$$

\* it's not quite true  
due to fast Strassen  
based schemes, but  
it's true enough

But if you know  $A$  is diagonal, now it's uncoupled,

$$\begin{bmatrix} & & \\ & \ddots & \\ & & \end{bmatrix} \begin{bmatrix} x \\ | \\ b \end{bmatrix}$$

so only  $\mathcal{O}(n)$  flops

If it's lower (or upper triangular), it's  $\mathcal{O}(n^2)$  flops

in Matlab/Sympy  
using a sparse  
structure will  
already make  
it pretty  
fast to solve

It turns out tridiagonal matrices can be solved

in  $\mathcal{O}(n)$  flops, e.g. "Thomas Algorithm"

or "Crout Factorization" (Algo 6.7 in our book)

## Theory

Thm 3.11 There is a unique natural cubic spline interpolant

proof we just proved this when we showed the above tridiagonal  
matrix is invertible

Thm 3.12 If  $f$  is differentiable at  $x_0$  and  $x_n$ , then there is

a unique clamped cubic spline interpolant

proof similar

(4b)

## Error Bounds

**Burden & Faires Thm 3.13** If  $f \in C^4([a,b])$  with  $\max_{x \in [a,b]} |f^{(4)}(x)| = M$  and

$s$  is the clamped cubic spline interpolant on

$$a = x_0 < x_1 < x_2 < \dots < x_n = b \quad \text{then}$$

$$(\forall x \in [a,b]) \quad |f(x) - s(x)| \leq M \cdot \frac{5}{384} \cdot \max_{0 \leq j \leq n-1} (x_{j+1} - x_j)^4.$$

(Similar results hold for natural and not-a-knot cubic splines)

## Interpretation of Thm 3.13

Consider uniformly spaced points

$$x_0 = a, \quad x_1 = a + h, \quad x_2 = a + 2h, \\ \dots, \quad x_n = a + nh = b$$

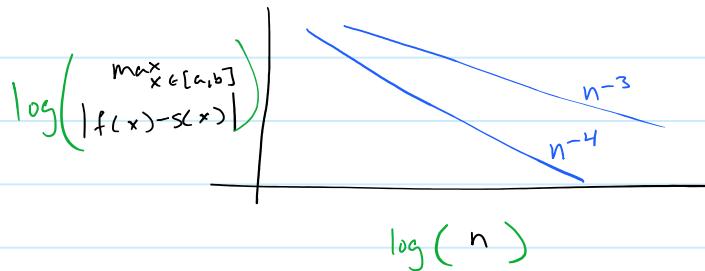
$$\text{so } h = \frac{b-a}{n}$$

then the theorem says the error is

$$|f(x) - s(x)| \leq \text{constant} \cdot h^4$$

$$\text{i.e. } \leq \text{constant} \cdot n^{-4}$$

We say it is 4<sup>th</sup> order accurate



$y = n^\alpha$  is a straight-line on a log-log plot

top 10 fact you should know after this course

## Similar Thm (Thm 5.3.1 Driscoll + Braun)

If  $f \in C^4[a,b]$  and  $s$  is the not-a-knot cubic Spline on  $n+1$  equispaced points (spacing is  $h = \frac{b-a}{n}$ ) then  $\|f - s\|_\infty = O(h^4)$

i.e. for  $h$  sufficiently small,  
 $\exists c \in \mathbb{R}, \forall x \in [a,b] \quad |f(x) - s(x)| \leq c \cdot h^4$

Summary

Cubic Splines are piecewise cubic,  $C^2$  functions that interpolate on the nodes, are 4<sup>th</sup> order accurate, and require  $O(n)$  computation for  $n+1$  nodes.