

7. Interpolation

- polynomial interpolation
- Newton divided difference
- Lagrange interpolation
- spline interpolation

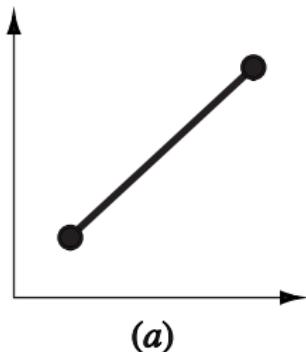
Polynomial interpolation

construct an n th-order polynomial

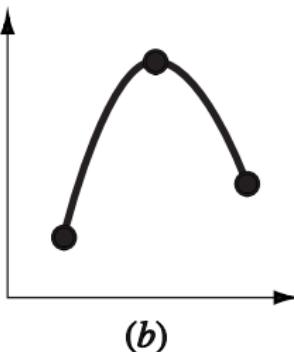
$$f_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

that passes *exactly* through the given data points

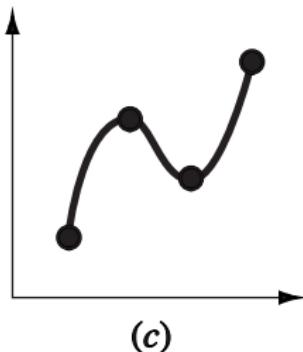
- n th order polynomial requires $n + 1$ data points
- examples: (a) $n = 1$ straight line; (b) $n = 2$ parabola; (c) $n = 3$



(a)



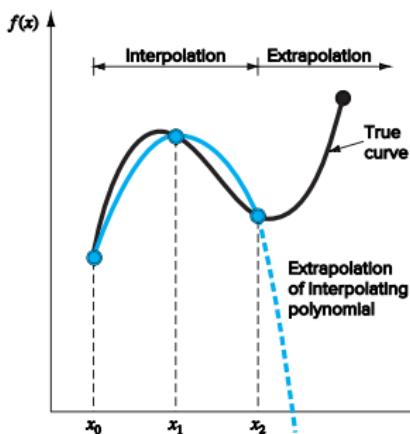
(b)



(c)

The need for interpolation

- building blocks for other, more complex algorithms in differentiation, integration, solution of differential equations, approximation theory, ...
 - e.g., finding approximations for derivatives and integrals of a complicated function
- used for prediction: provides a formula to estimate *intermediate values* x other than the available data, x_0, \dots, x_n
 - *interpolation*: x is inside the smallest interval containing all the data
 - *extrapolation*: x is outside that interval



Polynomial interpolation via linear equations

fit polynomial $f_n(x) = \sum_{k=0}^n a_k x^k$ to data $(x_0, f(x_0)), \dots, (x_n, f(x_n))$

- substitute $\{(x_i, f(x_i))\}_{i=0}^n$ in $f_n(x)$ yields $n + 1$ linear equations $Xa = f$:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}$$

coefficient matrix X known as *Vandermonde matrix*

- X is invertible for distinct $x_i \implies$ unique polynomial coefficient
- given three points $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$, substitute to obtain

$$\begin{aligned} f(x_0) &= a_0 + a_1 x_0 + a_2 x_0^2 && \begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{bmatrix} \\ f(x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 &\implies& \\ f(x_2) &= a_0 + a_1 x_2 + a_2 x_2^2 && \end{aligned}$$

Example

- fit a polynomial, $f_1(x) = a_0 + a_1x$, through $(1, 1)$ and $(2, 3)$

- the interpolating conditions are

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \Rightarrow a_0 = -1, \quad a_1 = 2 \Rightarrow f_1(x) = 2x - 1$$

- fit a polynomial $f_2(x) = a_0 + a_1x + a_2x^2$ through $(1, 1)$, $(2, 3)$, $(4, 3)$

- 3×3 linear system for the unknown coefficients c_j :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 16 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix} \Rightarrow a_0 = -\frac{7}{3}, \quad a_1 = 4, \quad a_2 = -\frac{2}{3}$$

- the desired interpolating polynomial is

$$f_2(x) = \frac{-2x^2 + 12x - 7}{3}$$

- for instance at $x = 3$, $f_2(3) = \frac{11}{3} = 3.6667$, which is lower than $f_1(3) = 5$

Efficiency and numerical cautions

Solving Vandemonde system

- forming and solving a full Vandermonde system is not the most efficient path
- Vandermonde matrices are ill-conditioned for large n or widely spaced x_i
- small data or rounding errors can produce large coefficient errors

Special algorithms

- specialized algorithms are faster and more stable
- two forms especially useful for computer implementation:
 1. *Newton polynomial*
 2. *Lagrange polynomial*
- allows forming $f_n(x)$ directly without solving for all a_k

Outline

- polynomial interpolation
- **Newton divided difference**
- Lagrange interpolation
- spline interpolation

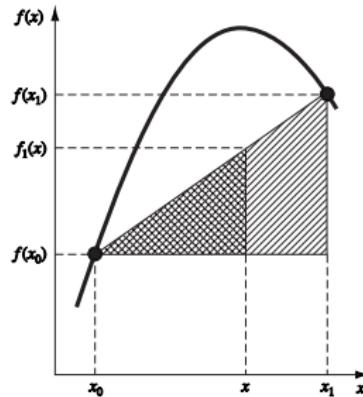
Linear interpolation (first-order)

- connect two data points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ with a straight line
- from similar triangles, we get the **linear interpolation formula**:

$$\frac{f_1(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

\Rightarrow

$$f_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$



- slope $f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$ is first *finite divided difference* of 1st-derivative
 - as $x_1 \rightarrow x_0$, first divided difference $f[x_1, x_0] \rightarrow f'(x_0)$ for smooth f
- as interval $[x_0, x_1]$ shrinks, f is better approximated by a straight line

Example

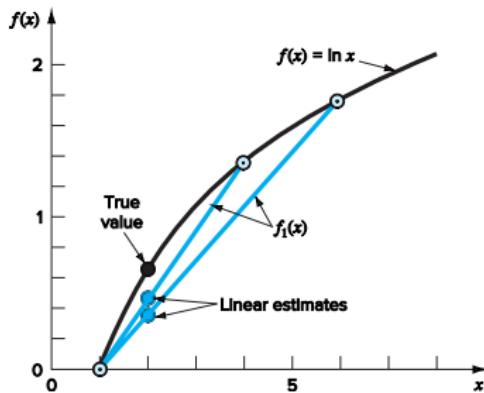
estimate $\ln 2 = 0.6931472$ on $[1, 6]$ and $[1, 4]$

(a) interpolate on $[1, 6]$: using $\ln 1 = 0$, $\ln 6 = 1.791759$,

$$f_1(2) = 0 + \frac{1.791759 - 0}{6 - 1}(2 - 1) = 0.3583519, \quad \varepsilon_t = 48.3\%$$

(b) interpolate on $[1, 4]$: using $\ln 1 = 0$, $\ln 4 = 1.386294$,

$$f_1(2) = 0 + \frac{1.386294 - 0}{4 - 1}(2 - 1) = 0.4620981, \quad \varepsilon_t = 33.3\%$$



Quadratic (parabola) interpolation (second-order)

to introduce *curvature*, use three points $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$

Newton form quadratic polynomial

$$f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

- equivalent to the standard quadratic $a_0 + a_1x + a_2x^2$ via collecting terms
- coefficients determined by enforcing exactness at x_0, x_1, x_2 :

$$b_0 = f(x_0), \quad b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$b_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

- b_1 is *first divided difference*, and b_2 is *second divided difference*

Example

use quadratic interpolation for $\ln 2$ given data

$$x_0 = 1, f(x_0) = 0; \quad x_1 = 4, f(x_1) = 1.386294; \quad x_2 = 6, f(x_2) = 1.791759$$

using previous formulas, we have

$$b_0 = f(x_0) = 0, \quad b_1 = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$

$$b_2 = \frac{\frac{1.791759 - 1.386294}{6 - 4} - 0.4620981}{6 - 1} = -0.0518731$$

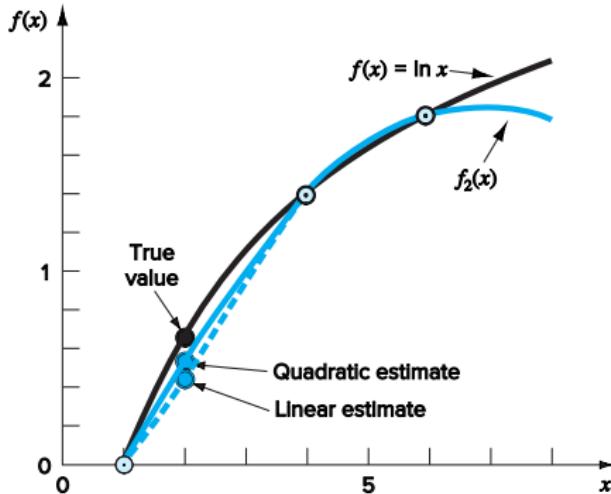
hence, the interpolant:

$$f_2(x) = 0 + 0.4620981(x - 1) - 0.0518731(x - 1)(x - 4)$$

and the estimated value at $x = 2$ is

$$f_2(2) = 0.5658444, \quad \varepsilon_t = \frac{0.6931472 - 0.5658444}{0.6931472} \times 100\% = 18.4\%$$

Example



- quadratic interpolant improves over linear case; added curvature reduces error
- linear: exact at x_0, x_1 ; error proportional to local curvature (second derivative)
- quadratic: exact at x_0, x_1, x_2 ; incorporates a second-order term via b_2 , thus better tracks smooth curvature of $\ln x$ near $x = 2$

Newton interpolating polynomial

Newton polynomial: n th-order Newton interpolating polynomial for $n + 1$ points:

$$f_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

- coefficients determined by enforcing exactness at x_0, x_1, \dots, x_n
- b_0 computed using only $f(x_0)$; b_1 computed using only $(x_0, f(x_0)), (x_1, f(x_1))$
- … b_n computed using $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$
- allows introducing interpolation data (x_i, y_i) one pair at a time

Coefficients in terms of finite divided differences

$$f_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

- coefficient b_j is j th finite divided differences:

$$b_0 = f(x_0), b_1 = f[x_1, x_0], b_2 = f[x_2, x_1, x_0], \dots, b_n = f[x_n, x_{n-1}, \dots, x_0]$$

- recursive definitions:

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j} \quad \text{1st finite divided difference}$$

⋮

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k} \quad \text{2nd finite divided difference}$$

⋮

$$f[x_n, \dots, x_0] = \frac{f[x_n, \dots, x_1] - f[x_{n-1}, \dots, x_0]}{x_n - x_0} \quad \text{nth finite divided difference}$$

Recursive computation of divided differences

- example structure:

I	x_I	$f(x_I)$	First	Second	Third
0	x_0	$f(x_0)$	$f[x_1, x_0]$	$f[x_2, x_1, x_0]$	$f[x_3, x_2, x_1, x_0]$
1	x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	
2	x_2	$f(x_2)$	$f[x_3, x_2]$		
3	x_3	$f(x_3)$			

- the divided difference coefficients satisfy the recursive formula

$$f[x_i, x_{i-1}, \dots, x_0] = \frac{f[x_i, \dots, x_1] - f[x_{i-1}, \dots, x_0]}{x_i - x_0}$$

require to compute for $0 \leq k < j \leq i \leq n$:

$$f[x_j] = f(x_j), \quad f[x_j, \dots, x_k] = \frac{f[x_j, \dots, x_{k+1}] - f[x_{j-1}, \dots, x_k]}{x_j - x_k}$$

- higher-order differences are computed from lower-order ones
- this recursive property is the basis of efficient computer algorithms

Example

extend last example to cubic interpolation for $\ln 2$ using points:

$$x_0 = 1, f(x_0) = 0; \quad x_1 = 4, f(x_1) = 1.386294$$

$$x_2 = 6, f(x_2) = 1.791759; \quad x_3 = 5, f(x_3) = 1.609438$$

i	x_i	$f[x_i]$	$f[x_{i+1}, x_i]$	$f[x_{i+2}, x_{i+1}, x_i]$	$f[x_3, x_2, x_1, x_0]$
0	1	0	0.4620981	-0.05187311	0.007865529
1	4	1.386294	0.2027326	-0.02041100	
2	6	1.791759	0.1823216		
3	5	1.609438			

for example

$$f[x_1, x_0] = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$

$$f[x_2, x_1, x_0] = \frac{0.2027326 - 0.4620981}{6 - 1} = -0.05187311$$

$$f[x_3, x_2, x_1, x_0] = \frac{-0.02041100 - (-0.05187311)}{5 - 1} = 0.007865529$$

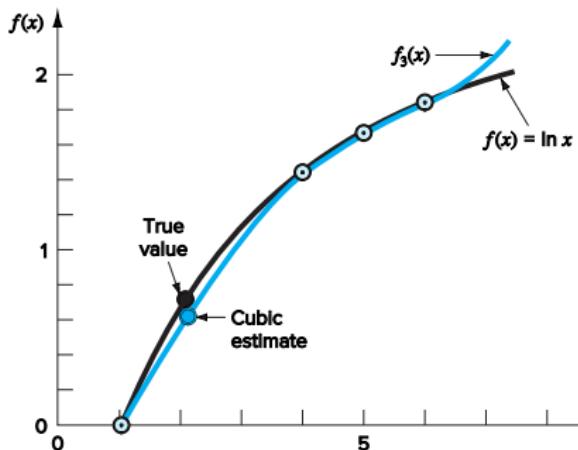
Example

resulting polynomial coefficient extracted from first row:

$$f_3(x) = 0 + 0.4620981(x - 1) - 0.05187311(x - 1)(x - 4) \\ + 0.007865529(x - 1)(x - 4)(x - 6)$$

estimate at $x = 2$ (true value $\ln 2 = 0.6931472$):

$$f_3(2) = 0.6287686, \quad \varepsilon_t = \frac{0.6931472 - 0.6287686}{0.6931472} \times 100\% = 9.3\%$$



Error of Newton interpolating polynomial

- structure of Newton polynomial resembles a Taylor series expansion

$$f_n(x) = f(x_0) + (x - x_0)f[x_1, x_0] + \dots$$

- if $f(x)$ is truly an n th-order polynomial, the interpolating polynomial is exact
- analogy with Taylor series remainder for n th order polynomial:

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x - x_0)(x - x_1)\cdots(x - x_n), \quad \xi \text{ lies within the data interval}$$

- practical difficulty: $f^{(n+1)}(\xi)$ is usually unknown
- alternative finite difference formulation

$$R_n = f[x, x_n, \dots, x_0](x - x_0)(x - x_1)\cdots(x - x_n)$$

- if an extra data point $f(x_{n+1})$ is available, error can be estimated as

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

Example: error estimation

estimate error of quadratic interpolant for $\ln 2$ using extra point $f(5) = 1.609438$

- from page 7.10, quadratic estimate $f_2(2) = 0.5658444$
- true error

$$E_t = 0.6931472 - 0.5658444 = 0.1273028$$

- from page 7.15, we have $f[x_3, x_2, x_1, x_0] = 0.007865529$
- error estimate:

$$\begin{aligned} R_2 &= f[x_3, x_2, x_1, x_0](x - 1)(x - 4)(x - 6) \\ &= 0.007865529(2 - 1)(2 - 4)(2 - 6) = 0.0629242 \end{aligned}$$

estimate is of the same order of magnitude as the true error

Outline

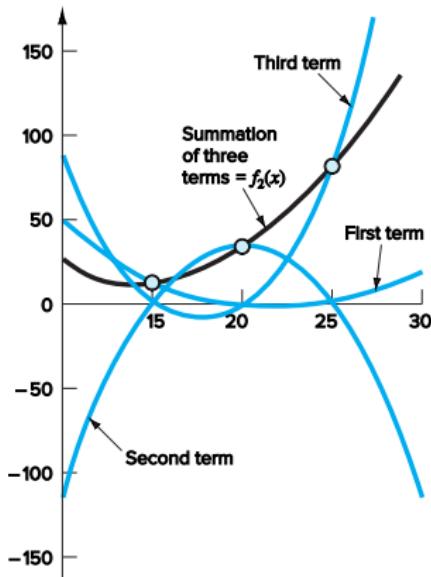
- polynomial interpolation
- Newton divided difference
- **Lagrange interpolation**
- spline interpolation

Lagrange interpolating polynomials

$$f_n(x) = \sum_{i=0}^n L_i(x)f(x_i)$$

where

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}$$



- a reformulation of Newton polynomial that avoids divided differences
- each $L_i(x)$ is 1 at $x = x_i$ and 0 at other sample points
- the sum is the unique n th-order polynomial that passes through all $n + 1$ points

First-, second-, third-order Lagrange polynomial

- first-order polynomial ($n = 1$):

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

- second-order polynomial ($n = 2$):

$$\begin{aligned} f_2(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) \\ &\quad + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2) \end{aligned}$$

- third-order polynomial ($n = 3$):

$$\begin{aligned} f_3(x) &= \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} f(x_0) + \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} f(x_1) \\ &\quad + \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} f(x_3) \end{aligned}$$

Example

use Lagrange first-order interpolation to estimate $\ln 2$ with data

$$x_0 = 1, f(x_0) = 0, x_1 = 4, f(x_1) = 1.386294$$

we have

$$f_1(2) = \frac{2 - 4}{1 - 4} \cdot 0 + \frac{2 - 1}{4 - 1} \cdot 1.386294 = 0.4620981$$

extending the example with three points

$$x_0 = 1, f(x_0) = 0, x_1 = 4, f(x_1) = 1.386294, x_2 = 6, f(x_2) = 1.791760$$

gives

$$\begin{aligned} f_2(2) &= \frac{(2 - 4)(2 - 6)}{(1 - 4)(1 - 6)} \cdot 0 + \frac{(2 - 1)(2 - 6)}{(4 - 1)(4 - 6)} \cdot 1.386294 \\ &\quad + \frac{(2 - 1)(2 - 4)}{(6 - 1)(6 - 4)} \cdot 1.791760 = 0.5658444 \end{aligned}$$

Observation: agrees with the Newton interpolation

Example: parachutist velocity interpolation problem

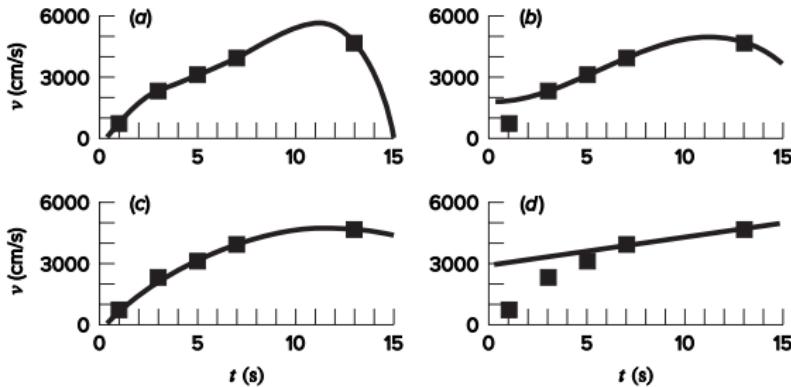
estimate $v(10)$ by polynomial interpolation of orders $n = 1, 2, 3, 4$ given data

t	1	3	5	7	13
v	800	2310	3090	3940	4755

- (a) quartic ($n = 4$): use all five points
- (b) cubic ($n = 3$): use $\{(3, 2310), (5, 3090), (7, 3940), (13, 4755)\}$
- (c) quadratic ($n = 2$): use $\{(5, 3090), (7, 3940), (13, 4755)\}$
- (d) linear ($n = 1$): use $\{(7, 3940), (13, 4755)\}$

note: selecting nearest neighbors typically improves stability and accuracy

Computed estimates at $t = 10$ s



- higher orders (cubic, quartic) *overshoot* trend between $t = 7$ and $t = 13$
- higher-degree polynomials are ill-conditioned and sensitive to data spacing/noise
- prefer *low-order* interpolation with *nearby* points for local estimates
- for noisy data, consider *regression* (least squares) rather than exact interpolation
- for many points over a range, consider *piecewise* low-order methods (e.g., splines)

Inverse interpolation

- inverse problem: given $f(x) = f^*$, determine x
- e.g., find x such that $f(x) = \frac{1}{x} = 0.3 \Rightarrow$ true value $x = 3.333$
- x values typically evenly spaced
- example: $f(x) = 1/x$

x	1	2	3	4	5	6	7
$f(x)$	1	0.5	0.3333	0.25	0.2	0.1667	0.1429

Naive approach: swap variables and interpolate x vs $f(x)$

$y = f(x)$	0.1429	0.1667	0.2	0.25	0.3333	0.5	1
$g(y) = x$	7	6	5	4	3	2	1

- but $f(x)$ values are nonuniform \Rightarrow abscissa “telescoped”
- this often causes oscillations even with low-order polynomials

Polynomial strategy for inverse interpolation

alternative approach:

- fit interpolating polynomial $f_n(x)$ to original data ($f(x)$ vs x)
- then solve $f_n(x) = f^*$ for x using root-finding methods

Example: fit quadratic through $(2, 0.5), (3, 0.3333), (4, 0.25)$

$$f_2(x) = 1.08333 - 0.375x + 0.041667x^2$$

solve for $f(x) = 0.3$:

$$0.3 = 1.08333 - 0.375x + 0.041667x^2$$

quadratic formula:

$$x = \frac{0.375 \pm \sqrt{(-0.375)^2 - 4(0.041667)(0.78333)}}{2(0.041667)}$$

roots: $x = 5.704, 3.296$

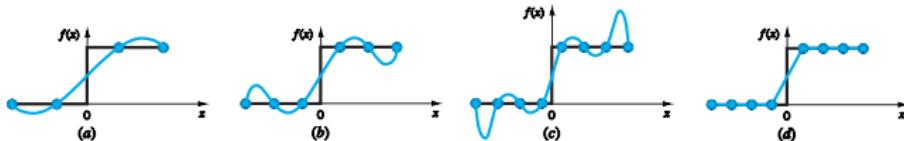
\Rightarrow choose $x = 3.296$ as approximation to true 3.333

Outline

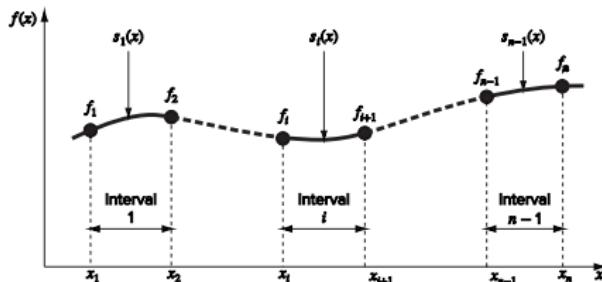
- polynomial interpolation
- Newton divided difference
- Lagrange interpolation
- **spline interpolation**

Spline interpolation

- apply lower-order polynomials to subsets of data points
- these connecting polynomials are called **spline functions**



- higher-order polynomials oscillate wildly near abrupt changes
- splines (limited to lower-order curves) minimize oscillations
- n data points $\Rightarrow n - 1$ intervals; each interval i has spline $s_i(x)$



data points where two splines meet are called *knots* or *break points*

Linear splines

- straight line between two adjacent points $[x_i, x_{i+1}], i = 1, \dots, n - 1$
- called *piecewise linear* or *broken line* interpolation

Linear spline: for interval $[x_i, x_{i+1}]$, use Newton first-order polynomial

$$s_i(x) = a_i + b_i(x - x_i), \quad a_i = f_i, \quad b_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$$

therefore,

$$s_i(x) = f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}(x - x_i), \quad \text{for } x \in [x_i, x_{i+1}]$$

- n data points $\Rightarrow n - 1$ intervals
- each $s_i(x)$ can be used to evaluate f between (x_i, x_{i+1})
- resulting interpolant is continuous but not differentiable at knots

Example

fit the data in table with first-order splines and evaluate the function at $x = 5$

i	x_i	f_i
1	3.0	2.5
2	4.5	1.0
3	7.0	2.5
4	9.0	0.5

- 4 data points \Rightarrow 3 intervals
- spline in interval i :

$$s_i(x) = f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}(x - x_i)$$

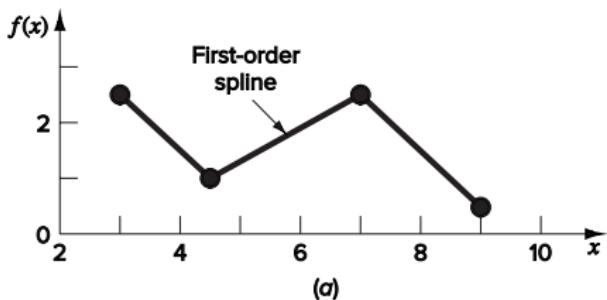
- for instance, the second interval ($x = 4.5$ to $x = 7$) spline is

$$s_2(x) = 1.0 + \frac{2.5 - 1.0}{7.0 - 4.5}(x - 4.5)$$

evaluating at $x = 5$:

$$s_2(5) = 1.0 + \frac{2.5 - 1.0}{7.0 - 4.5}(5 - 4.5) = 1.3$$

Example



Remarks

- first-order splines connect data with straight lines
- disadvantage: not smooth at knots (slope changes abruptly)
- first derivative is discontinuous at joining points
- higher-order splines overcome this by enforcing derivative continuity
- spline of at least $n + 1$ order need to ensure n th derivatives are continuous at knots

Quadratic splines

for n data points (x_i, f_i) , construct *piecewise quadratics* each interval $[x_i, x_{i+1}]$:

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2$$

- for n data points $\Rightarrow n - 1$ intervals $\Rightarrow 3(n - 1)$ unknowns (a_i, b_i, c_i)
- need $3(n - 1)$ conditions to solve system
- quadratic splines require continuous first derivatives at knots

Conditions for quadratic splines

Pass through data

$$s_i(x_i) = f_i \Rightarrow f_i = a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2 \Rightarrow a_i = f_i$$

so

$$s_i(x) = f_i + b_i(x - x_i) + c_i(x - x_i)^2$$

reduces unknowns to $2(n - 1)$

Function continuity at knots: define $h_i = x_{i+1} - x_i$, then

$$s_i(x_{i+1}) = f_{i+1} \Rightarrow f_i + b_i h_i + c_i h_i^2 = f_{i+1}, \quad i = 1, \dots, n - 1$$

Derivative continuity at interior knots

$$s'_i(x_{i+1}) = s'_{i+1}(x_{i+1}) \Rightarrow b_i + 2c_i h_i = b_{i+1}, \quad i = 1, \dots, n - 2$$

Zero second derivative at first point

$$c_1 = 0$$

this implies first two points will be connected by a straight line

Example

fit quadratic splines to the data

i	x_i	f_i
1	3.0	2.5
2	4.5	1.0
3	7.0	2.5
4	9.0	0.5

use the results to estimate the value of the function at $x = 5$

- for four data points ($n = 4$), we have $n - 1 = 3$ intervals
- after continuity condition and zero 2nd-derivative condition ($c_1 = 0$), we need

$$2(4 - 1) - 1 = 5$$

conditions

Example

continuity at knots yields (with $c_1 = 0$)

$$f_1 + b_1 h_1 = f_2$$

$$f_2 + b_2 h_2 + c_2 h_2^2 = f_3$$

$$f_3 + b_3 h_3 + c_3 h_3^2 = f_4$$

derivative continuity conditions (with $c_1 = 0$)

$$b_1 = b_2$$

$$b_2 + 2c_2 h_2 = b_3$$

function and interval widths $h_1 = 1.5$, $h_2 = 2.5$, $h_3 = 2.0$

putting things together, results in the system of linear equations:

$$\begin{bmatrix} 1.5 & 0 & 0 & 0 & 0 \\ 0 & 2.5 & 6.25 & 0 & 0 \\ 0 & 0 & 0 & 2 & 4 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 5 & -1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ c_2 \\ b_3 \\ c_3 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \\ -2 \\ 0 \\ 0 \end{bmatrix}$$

Example

solution is

$$b_1 = -1, \quad b_2 = -1, \quad c_2 = 0.64, \quad b_3 = 2.2, \quad c_3 = -1.6$$

and the quadratic splines are

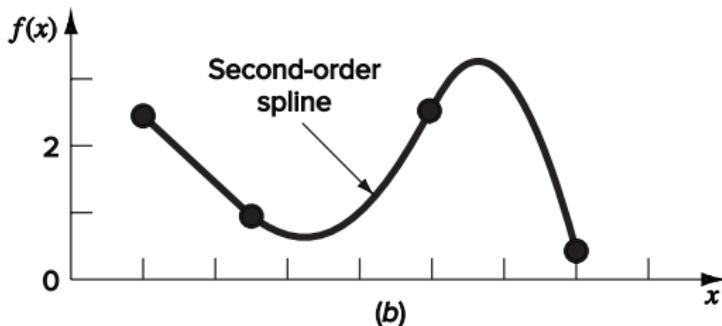
$$s_1(x) = 2.5 - (x - 3)$$

$$s_2(x) = 1.0 - (x - 4.5) + 0.64(x - 4.5)^2$$

$$s_3(x) = 2.5 + 2.2(x - 7.0) - 1.6(x - 7.0)^2$$

so, our estimate at $x = 5$ is

$$s_2(5) = 1.0 - (0.5) + 0.64(0.5^2) = 0.66$$



Cubic splines

- linear and quadratic splines lack smoothness or symmetry
- cubic splines are most commonly used
- require continuous 1st and 2nd derivatives at knots
- goal: for n data points (x_i, f_i) , construct *piecewise cubics* on $n - 1$ intervals

$$[x_i, x_{i+1}]$$

Cubic splines: on interval i , use

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

unknowns per interval: $a_i, b_i, c_i, d_i \Rightarrow$ total $4(n - 1)$ unknowns

Cubic spline interpolation

on $[x_i, x_{i+1}]$,

$$s_i(x) = f_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

-
1. solve tridiagonal system for c_1, \dots, c_n :

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & & \\ \ddots & \ddots & \ddots & & \\ 0 & \cdots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3(f[x_3, x_2] - f[x_2, x_1]) \\ \vdots \\ 3(f[x_n, x_{n-1}] - f[x_{n-1}, x_{n-2}]) \\ 0 \end{bmatrix}$$

2. back-substitution for remaining coefficients: for $i = 1, \dots, n - 1$

$$b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3} (2c_i + c_{i+1})$$

$$d_i = \frac{c_{i+1} - c_i}{3h_i}$$

Derivation

Pass through data: at $x = x_i$,

$$f_i = a_i \Rightarrow a_i = f_i$$

$$\text{so } s_i(x) = f_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Function continuity at knots: with $h_i = x_{i+1} - x_i$

$$f_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_{i+1}$$

First-derivative continuity

$$s'_i(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2$$

at x_{i+1} ,

$$b_i + 2c_i h_i + 3d_i h_i^2 = b_{i+1}$$

Second-derivative continuity

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

at x_{i+1} ,

$$c_i + 3d_i h_i = c_{i+1}$$

Derivation

eliminate d_i :

$$d_i = \frac{c_{i+1} - c_i}{3h_i}$$

substitute back into first two equations:

$$f_i + b_i h_i + \frac{h_i^2}{3} (2c_i + c_{i+1}) = f_{i+1}$$

$$b_{i+1} = b_i + h_i(c_i + c_{i+1}) \implies b_i = b_{i-1} + h_{i-1}(c_{i-1} + c_i)$$

solve first equation for b_i and shift index:

$$b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3} (2c_i + c_{i+1})$$

$$b_{i-1} = \frac{f_i - f_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{3} (2c_{i-1} + c_i)$$

we now substitute these two equations into

$$b_i = b_{i-1} + h_{i-1}(c_{i-1} + c_i)$$

Derivation

putting things together yields a relation for c :

$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_ic_{i+1} = 3\left(\frac{f_{i+1} - f_i}{h_i} - \frac{f_i - f_{i-1}}{h_{i-1}}\right)$$

or with divided differences notation $f[x_i, x_j] = \frac{f_j - f_i}{x_j - x_i}$,

$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_ic_{i+1} = 3(f[x_{i+1}, x_i] - f[x_i, x_{i-1}])$$

in matrix form, this yields a tridiagonal system of linear equations

Natural end conditions (straight at ends): 2nd derivatives vanish at the endpoints:

$$s_1''(x_1) = 0 \Rightarrow c_1 = 0$$

$$s_{n-1}''(x_n) = 0 \Rightarrow c_{n-1} + 3d_{n-1}h_{n-1} = c_n = 0$$

where we introduced an extraneous parameter c_n

Example

fit cubic splines to the data

i	x_i	f_i
1	3.0	2.5
2	4.5	1.0
3	7.0	2.5
4	9.0	0.5

use the results to estimate the value of the function at $x = 5$

tridiagonal system of equations

$$\begin{bmatrix} 1 & & & \\ h_1 & 2(h_1 + h_2) & h_2 & \\ & h_2 & 2(h_2 + h_3) & h_3 \\ & & & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3(f[x_3, x_2] - f[x_2, x_1]) \\ 3(f[x_4, x_3] - f[x_3, x_2]) \\ 0 \end{bmatrix}$$

data values:

$$h_1 = 4.5 - 3.0 = 1.5, \quad h_2 = 7.0 - 4.5 = 2.5, \quad h_3 = 9.0 - 7.0 = 2.0$$

Example

matrix system becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1.5 & 8 & 2.5 & 0 \\ 0 & 2.5 & 9 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 4.8 \\ -4.8 \\ 0 \end{bmatrix}$$

- solution: $c_1 = 0, c_2 = 0.8395, c_3 = -0.7665, c_4 = 0$
- compute $b_i = \frac{f_{i+1}-f_i}{h_i} - \frac{h_i}{3} (2c_i + c_{i+1})$

$$b_1 = -1.4198, \quad b_2 = -0.1605, \quad b_3 = 0.0221$$

- compute $d_i = \frac{c_{i+1}-c_i}{3h_i} :$

$$d_1 = 0.1866, \quad d_2 = -0.2141, \quad d_3 = 0.1278$$

Example

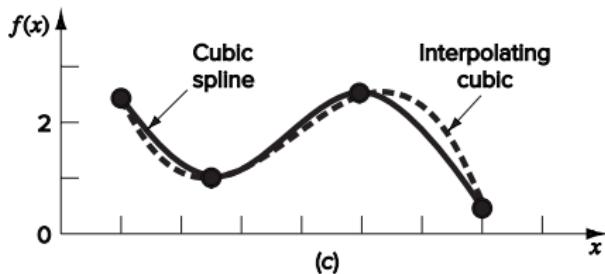
final cubic splines

$$s_1(x) = 2.5 - 1.4198(x - 3) + 0.1866(x - 3)^3$$

$$s_2(x) = 1.0 - 0.1605(x - 4.5) + 0.8395(x - 4.5)^2 - 0.2141(x - 4.5)^3$$

$$s_3(x) = 2.5 + 0.0221(x - 7) - 0.7665(x - 7)^2 + 0.1278(x - 7)^3$$

estimate at $x = 5$ (interval 2): $s_2(5) = 1.103$



cubic spline fit shows smoother and more accurate behavior than linear/quadratic

End conditions for cubic splines

Natural: $c_1 = 0, c_n = 0$ (spline straightens at endpoints)

Clamped: specify first derivatives at first and last nodes

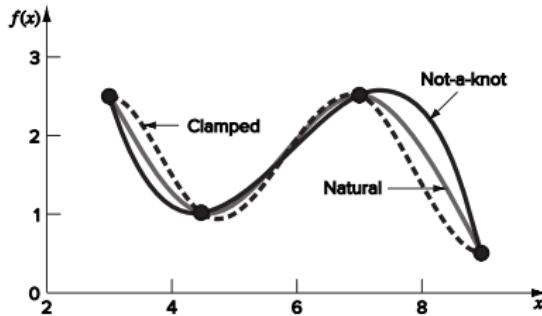
$$2h_1c_1 + h_1c_2 = 3f[x_2, x_1] - 3f'_1$$

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3f'_n - 3f[x_n, x_{n-1}]$$

Not-a-knot: enforce third derivative continuity at 2nd and next-to-last knots

$$h_2c_1 - (h_1 + h_2)c_2 + h_1c_3 = 0$$

$$h_{n-1}c_{n-2} - (h_{n-2} + h_{n-1})c_{n-1} + h_{n-2}c_n = 0$$



References and further readings

- S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers* (8th edition). McGraw Hill, 2021. (Ch.18)
- S. C. Chapra. *Applied Numerical Methods with MATLAB for Engineers and Scientists* (5th edition). McGraw Hill, 2023. (Ch.17, 18)