MA 668-Numerical Analysis I
Polynomial Interpolation

Dr. Mohebujjaman

Department of Mathematics
University of Alabama at Birmingham (UAB)

February 19, 2024

General Approximation and Interpolation

- Interpolation is a special case of approximation

Discrete and continuous approximation in one dimension:

- **Data fitting** (Discrete approximation problem)
  - Given a set of data points $\{(x_i, y_i)\},\ i = 0, 1, \cdots, n$ (without noise) $x_i \neq x_j,\ \ i \neq j$
  - Find a *reasonable* function $v(x)$ that fits the data points
  - Use $v(x)$ to interpolate data, satisfying

$$v(x_i) = y_i \ i = 0, 1, \cdots, n$$

- **Approximate functions**
  - $f(x)$: Complicated
  - Find simpler function $v(x) \approx f(x)$

$v(x)$

- Easy to evaluate and manipulate
- Reasonable

Need for interpolation

Why do we want to find an approximation function $v(x)$ in general?

- Prediction:
  - **Interpolation:** x inside the data abscissae
  - **Extrapolation:** x outside the data abscissae
- Manipulation: e.g., approximation of derivatives and integrals

Interpolants and their representation

- We generally assume **linear form** of interpolating function $v(x)$
- **# data points=# basis** functions, otherwise, least square sense. The resulting linear system

$$\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & \vdots & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_n(x_n) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- **Polynomial interpolation:** Monomial basis
- **Piecewise polynomial interpolation:** Polynomial interpolation in "piecces" rather than on the entire given interval
- **Trigonometric interpolation:** (Extremely useful in uncertainty quantification, and signal processing etc)

$$\phi_j(x) = \cos(jx), \; j = 0, 1, \cdots, n$$

Interpolants and their representation

1. **Constructing** the interpolant: Find c_j

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

2. **Evaluating** at a given x

Polynomial interpolation

- Easy to construct and evaluate
- Easy to sum, multiply, differentiate, and integrate (results are also polynomial)
- have widely varying characteristics despite their simplicity

Monomial interpolation:

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

Example: Fit a polynomial of degree at most n.

1. Let $n = 1$, and $(1, 1)$, and $(2, 3)$. Find $p_1(3) =?$
2. Let $n = 2$, and $(1, 1)$, $(2, 3)$, and $(4, 3)$. Find $p_2(3) =?$

Unique interpolating polynomial

$n + 1$ data points. Find $c_0, c_1, \cdots, c_n$

$$\underbrace{\begin{pmatrix} 1 & x_0^1 & x_0^2 & \cdots & x_0^n \\ 1 & x_1^1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^1 & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{X} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

X : Vandermonde matrix.

$$det(X) = \prod_{i=0}^{n-1} \left(\prod_{j=i+1}^{n} (x_j - x_i \right) \neq 0$$

since $x_j \neq x_i, \quad i \neq j. \implies$ Unique solution $\implies$ Unique interpolating polynomial. No matter which method or basis is used to obtain the interpolating polynomial.

Monomial basis for constructing interpolants

Pros:

- Intuitive simplicity
- Straightforwardness

Cons:

- Lake of stability: Small change in the problem, c_j may change completely
- Vandermonde matrix is often ill-conditioned, thus c_j maybe inaccurate
- $O(\frac{2}{3}n^3)$ flops for the Gaussian elimination in construction stage
- Evaluation stage requires $O(n^2)$; Nested form $O(n)$

Lagrange interpolation

Given $(x_i, y_i), \quad i = 0, 1, \cdots, n$

WTF: $\phi_j \ni c_j = y_j$ given $p(x) = p_n(x) = \sum_{j=0}^{n} c_j \phi_j(x)$.

Lagrange Polynomials of degree n, $L_j(x)$:

$$L_j(x_i) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

Unique polynomial of degree n:

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

$$p(x_i) = y_i$$

Lagrange interpolation

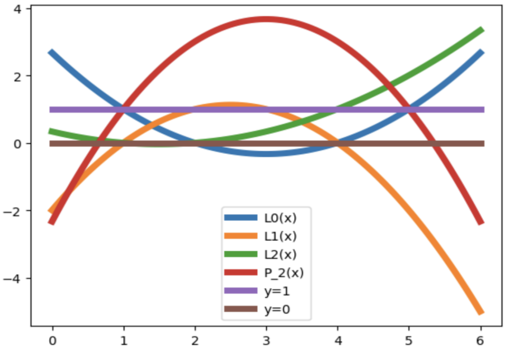
Example: Use $(1,1)$, $(2,3)$, and $(4,3)$, to construct Lagrange polynomials.

- $L_0(x) =$
- $L_1(x) =$
- $L_2(x) =$

Then, find:

- the interpolating polynomial $p_2(x) =?$
- $p_2(3) =?$

Lagrange interpolation



Properties of Lagrange polynomials

$$L_j(x) = \frac{(x-x_0)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)(x_j-x_1)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)}$$
$$= \prod_{\substack{i=0\\ i\neq j}}^{n} \frac{x-x_i}{x_j-x_i}$$

- $L_j(x_j) = 1, \quad j = 0, 1, \cdots, n$
- Form an ideally conditioned basis $\phi_j(x) = L_j(x)$
- $I\boldsymbol{c} = \boldsymbol{y} \implies c_j = y_j, \quad j = 0, 1, \cdots, n$
- L_j has n zeros and thus $n-1$ extrema
- $\sum_{j=0}^{n} L_j(x) = ?$

Lagrange Polynomial Interpolation

1. **Construction:** Given data $\{(x_i, y_i)\}_{i=0}^n$. Compute barycentric weights $w_j = \frac{1}{\prod\limits_{\substack{i=0\\ i\neq j}}^{n}(x_j - x_i)}$, and the quantities $w_j y_j,\ j = 0, 1, \cdots, n$.
2. **Evaluation:** Given an evaluation point $x \notin \{x_i\}_{i=0}^n$, compute

$$p(x) = \frac{\sum\limits_{j=0}^{n} \frac{w_j y_j}{(x - x_j)}}{\sum\limits_{j=0}^{n} \frac{w_j}{(x - x_j)}}$$

- Construction cost of **barycentric weights** w_j: $O(n^2)$
- **Evaluation stage:** $\psi(x) = \prod\limits_{i=0}^{n}(x - x_i)$, $p(x) = \psi(x) \sum\limits_{j=0}^{n} \frac{w_j y_j}{(x - x_j)}$: $O(5n)$

Divided Difference and Newton's form

Motivation:

- Introduce interpolation data (x_i, y_i) one pair at a time, rather than all at once from the start
- Estimating error in the interpolating approximation
- Monomial basis: $\{\phi_j(x) = x^j\}_{j=0}^{n}$
  - construction stage costly
  - easy evaluation procedure
- Lagrange polynomial basis: $\left\{\phi_j(x) = \prod_{\substack{i=0 \\ i\neq j}}^{n} \frac{(x-x_i)}{(x_j-x_i)}\right\}_{j=0}^{n}$
  - construction stage is easy
  - evaluation of $p_n(x)$ is relatively involved

Newton polynomial basis

Newton polynomial basis can be viewed as a useful compromise:

$$\phi_j(x) := \prod_{i=0}^{j-1}(x - x_i), \quad j = 0, 1, \cdots, n.$$

Example: For a quadratic interpolant, we have

$$\phi_0(x) = 1,\ \phi_1(x) = x - x_0,\ \phi_2(x) = (x - x_0)(x - x_1).$$

Example: Use $(1, 1)$, $(2, 3)$, and $(4, 3)$, to construct Newton's polynomial basis, and find $p_2(x)$, and $p_2(3) = ?$

Remark

- *To evaluate c_0, we only need first data point*
- *To evaluate c_1, we only need the first two points*

Existence of an adaptive interpolant

Features of Newton representation:

- It is evolutionary
- Determine $p_{n-1}(x)$ from n data points, use it to cheaply construct $p_n(x)$ using (x_n, y_n)

Polynomial takes the following form:

$$\begin{aligned} p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) \\ + \cdots + c_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned}$$

- Is it always possible to determine c_j?
- If yes, in this particular form unique?

Existence of an adaptive interpolant

$\phi_j(x_i) = 0$, for $i = 0, 1, \cdots, j-1$, $\phi_j(x_j) \neq 0$.

$$\underbrace{\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & \vdots & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_n(x_n) \end{pmatrix}}_{A} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- A: Lower triangular
- $A_{ii} \neq 0$
- A: Non-singular $\implies$ uniqueness

Representation in terms of divided differences

$$p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}).$$

Let $y_i = f(x_i)$

1. c_0 ? $\ni \quad p_n(x_0) = f(x_0) \implies c_0 = f(x_0)$
2. c_1 ? $\ni \quad p_n(x_1) = f(x_1) \implies c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$
3. c_2 ? $\ni \quad p_n(x_2) = f(x_2) \underbrace{\implies}_{?} c_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}$
4. keep continue until all c_j are being found

c_j : j^{th} divided difference, denoted $f[x_0, x_1, \cdots, x_j]$.

Representation in terms of divided differences

c_j : j^{th} divided difference coefficient, denoted $f[x_0, x_1, \cdots, x_j]$.

1. $f[x_0] = c_0$
2. $f[x_0, x_1] = c_1$
3. $f[x_0, x_1, x_2] = c_2, \cdots, f[x_0, x_1, x_2, \cdots, x_n] = c_n$

Newton divided difference interpolation formula:

$$\begin{aligned} p_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + \cdots + f[x_0, x_1, x_2, \cdots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\ &= \sum_{j=0}^{n} \left(f[x_0, x_1, \cdots, x_j] \prod_{i=0}^{j-1} (x - x_i) \right) \end{aligned}$$

Recursive formula

$$f[x_0, x_1, x_2, \cdots, x_j] = \frac{f[x_1, x_2, \cdots, x_j] - f[x_0, x_1, \cdots, x_{j-1}]}{x_j - x_0}$$

Divided Differences

Given points $x_0, x_1, \cdots, x_n$, for arbitrary indices $0 \leq i < j \leq n$

1. $f[x_i] = f(x_i)$
2. $f[x_i, \cdots, x_j] = \frac{f[x_{i+1}, \cdots, x_j] - f[x_i, \cdots, x_{j-1}]}{x_j - x_i}$

Divided Difference Table:

i	x_i	$f[x_i]$	$f[x_{i-1}, x_i]$	$f[x_{i-2}, x_{i-1}, x_i]$	$\cdots$	$f[x_{i-n}, \cdots, x_i]$
0	x_0	$f(x_0)$				
1	x_1	$f(x_1)$	$\frac{f[x_1]-f[x_0]}{x_1-x_0}$			
2	x_2	$f(x_2)$	$\frac{f[x_2]-f[x_1]}{x_2-x_1}$	$f[x_0, x_1, x_2]$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
n	x_n	$f(x_n)$	$\frac{f[x_n]-f[x_{n-1}]}{x_n-x_{n-1}}$	$f[x_{n-2}, x_{n-1}, x_n]$	$\cdots$	$f[x_0, x_1, \cdots, x_n]$

- Construction cost: # divisions $\frac{n^2}{2}$, # addition n^2

Divided Differences

Example:

- Use $(1,1)$, $(2,3)$, and $(4,3)$, to construct Divided differences, and the interpolating polynomial $p_2(x)$.
- Then, add another point (5,4), and compute the $p_3(x)$

The process of adding just one more data point $(x_{n+1}, f(x_{n+1}))$ to an existing interpolant p_n of the first $n+1$ data points is

$$p_{n+1}(x) = p_n(x) + f[x_0, x_1, \cdots, x_{n+1}] \prod_{i=0}^{n} (x - x_i)$$

- Evaluation cost: Nested evaluation $2n$

Algorithm Comparison

Basis name	$\phi_j(x)$	Const.	Eval.	Feature
Monomial	x^j	$\frac{2}{3}n^3$	$2n$	Simple
Lagrange	$\prod_{\substack{i=0 \\ i\neq j}}^{n} \frac{(x-x_i)}{(x_j-x_i)}$	n^2	$5n$	most stable
Newton	$\prod_{i=0}^{j-1}(x-x_i)$	$\frac{3}{2}n^2$	$2n$	adaptive

Divided difference and derivatives

$$f[z_0, z_1] = \frac{f(z_1) - f(z_0)}{z_1 - z_0} \overbrace{=}^{MVT} f'(\xi)$$

for some ξ between z_0, and z_1.

Theorem (Divided Difference and Derivative)

Let the function f be defined and have k bounded derivatives in an interval $[a, b]$ and let $z_0, z_1, \cdots, z_k$ be $k+1$ distinct points in $[a, b]$. Then, there is a point $\xi \in [a, b]$ such that

$$f[z_0, z_1, \cdots, z_k] = \frac{f^{(k)}(\xi)}{k!}.$$

Error in polynomial interpolation

If p_n interpolates f at the $n+1$ points $x_0, x_1, \cdots, x_n$, and f has $n+1$ bounded derivatives on an interval $[a, b]$ containing the points, then for each $x \in [a, b]$ there is a point $\xi = \xi(x) \in [a, b]$ such that

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

Furthermore, we have the following error bound

$$\max_{a \le x \le b} |f(x) - p_n(x)| \le \frac{1}{(n+1)!} \max_{a \le t \le b} |f^{(n+1)}(t)| \max_{a \le s \le b} \prod_{i=0}^{n} |s - x_i|$$

which is independent of the basis used for the interpolating polynomial

Error in polynomial interpolation

- f is usually unknown so its derivatives
  - Compute $p_k(x), p_{k+1}(x), \cdots$ using different subsets of the points and compare results how well they agree

Example: The following maximum daily temperature (C) were recorded by every third day during of a month:

Day	3	6	9	12	15	18	21	24	27
Tem (C)	31.2	32.0	35.3	34.1	35.0	35.5	34.1	35.1	36

Estimate the maximum temperature at day $x = 13$ of that month with

- $x_0 = 9, x_1 = 12, x_2 = 15$, and $x_3 = 18$: cubic interpolation, $p_3(13) = 34.29$
- $x_0 = 12$, and $x_1 = 15$: linear interpolation, $p_1(13) = 34.4$

Motivation: Want good quality interpolation. Given

- smooth function $f(x)$, $x \in [a, b]$
- we are free to choose the $n+1$ abscissae, $x_0, x_1, \cdots, x_n$.
- how should we choose these points?
- $$\max_{a \le x \le b} |f(x) - p_n(x)| \le \frac{1}{(n+1)!} \max_{a \le t \le b} |f^{(n+1)}(t)| \max_{a \le s \le b} \prod_{i=0}^{n} |s - x_i|$$
- $$\min \max_{a \le s \le b} \prod_{i=0}^{n} |s - x_i|$$
- Chebyshev points: $x_0, x_1, \cdots, x_n$

Chebyshev points $x_i \in [-1, 1]$

$$x_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right),\ \ i = 0, 1, \cdots, n$$

For general interval $[a, b]$, mapping:

$$x = a + \frac{b-1}{2}(t+1), \quad t \in [-1, 1]$$

Chebyshev points $x_i \in [-1, 1]$

Interpolation error using Chebyshev points:

$$\min_{x_0, x_1, \cdots, x_n} \max_{-1 \le s \le 1} \prod_{i=0}^{n} |s - x_i|$$

For monic Chebyshev polynomial:

$$\min_{x_0, x_1, \cdots, x_n} \max_{-1 \le s \le 1} \prod_{i=0}^{n} |s - x_i| = 2^{-n}$$

Interpolation error bound in this case:

$$\max_{a \le x \le b} |f(x) - p_n(x)| \le \frac{1}{(n+1)! 2^n} \max_{a \le t \le b} |f^{(n+1)}(t)|$$

Experiment on Runge function

$$f(x) = \frac{1}{1+25x^2}, \quad -1 \le x \le 1$$

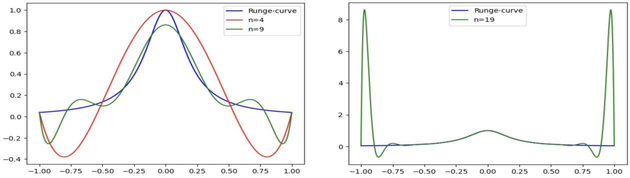


Figure: Equally spaced abscissae

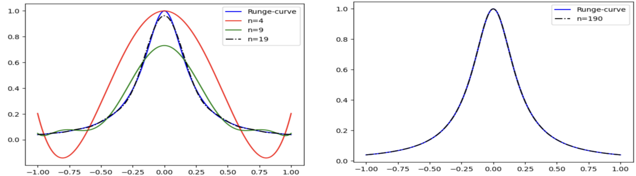


Figure: Chebyshev points as abscissae

Piecewise Polynomial Interpolation

Motivation: We want to build robust interpolation methods so that it works even if

- number of data points is large
- their abscissae locations are not under our control
- interval in which function is approximated is long

Piecewise Polynomial Interpolation

Shortcomings of polynomial interpolation:

- The error term

$$e_n(x) := f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

  may not be small if $\frac{\|f^{(n+1)}\|}{(n+1)!}$ isn't.
- High order polynomials tend to oscillate "unreasonably".
- Data often are only piecewise smooth, whereas polynomials are infinitely differentiable. The higher derivative $f^{(n+1)}$ may blow up (or very large) in such a case.
- Changing any one data value may drastically alter the entire interpolant.

Piecewise Polynomials

To reduce error without increasing the degree n, we reduce the size $b - a$.

Remark

Simply rescaling the independent variable x will not help!

- partition: $a = t_0 < t_1 < \cdots < t_r = b$
- use a (relatively low degree) polynomial interpolation $s_i(x)$ for each $[t_i, t_{i+1}],\ i = 0, 1, \cdots, r-1$
- these $s_i(x)$ are then patched together to form a global interpolating curve $v(x) \in C^1$ or C^2 so that

$$v(x) = s_i(x), \quad t_i \le x \le t_{i+1},\ i = 0, 1, \cdots, r-1$$

- The points $t_0, t_1, \cdots, t_r$ are called *break points*.

Broken Line Interpolation

- Polynomial pieces are linear
- Piecewise linear interpolant is continuous (but not continousely differentiable) everywhere

Data points: $(x_i, y_i),\ i = 0, 1, \cdots, 5$
Newton's formula for a linear polynomial interpolant

$$v(x) = s_i(x) = f(x_i) + f[x_i, x_{i+1}](x - x_i), \quad x_i \le x \le x_{i+1},\ i = \overline{0,4}$$

Advantages:

- Simple
- Maximum and miminum values are at the data points

Disadvantage:

- Not smooth enough
- Discontinous first derivative

Error bound for piecewise linear interpolation

- partition: $a = t_0 < t_1 < \cdots < t_n = b$
- $h = \max\limits_{1 \le i \le n} (t_i - t_{i-1})$
- General Interpolation Error

$$e_n(x) := f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

- Error bound for piecewise linear interpolation

$$|f(x) - v(x)| \le \frac{h^2}{8} \max_{a \le \xi \le b} |f''(\xi)|$$

- At least for equally spaced data points, as $h \downarrow n \uparrow$, $e_n(x) \downarrow$, $O(h^2) = O(n^{-2})$

Other piecewise polynomial interpolation

Piecewise constant interpolation:

- $t_i = \frac{x_i + x_{i-1}}{2}, \quad i = 1, 2, \cdots, n$
- $v(x) = s_i(x) = f(x_i), \quad t_i \leq x < t_{i+1}$
- $O(h)$

Piecewise cubic interpolation: To have higher smoothness, say, C^1 or C^2, we must increase the degree of each polynomial piece. $t_i = x_i$

$$v(x) = s_i(x) = a_i + b_i(x - t_i) + c_i(x - t_i)^2 + d_i(x - t_i)^3,$$

$$t_i \leq x \leq t_{i+1}, \; i = 0, \cdots, r - 1$$

Piecewise cubic interpolation

- Unknown: $4r$. Require $4r$ algebraic conditions.
  - Interpolation conditions
  - Continuity conditions
  - $s_i(t_i) = f(t_i),\ s_i(t_{i+1}) = f(t_{i+1}),\ \ i = 0, 1, \cdots, r-1.$ $2r$ conditions
  - $s_i^{'}(t_i) = f^{'}(t_i), s_i^{'}(t_{i+1}) = f^{'}(t_{i+1}).$ $2r$ conditions

which is called the **piecewise cubic Hermite interpolation**.

Note: Continuity is implied by

$$s_i(t_{i+1}) = f(t_{i+1}) = s_{i+1}(t_{i+1})$$

Piecewise Polynomial Interpolation Error

Let v interpolate f at the $n+1$ points $x_0 < x_1 < \cdots < x_n$, $h = \max\limits_{1\le i\le n} x_i - x_{i-1}$. Then for each $x \in [a,b]$ containing the above points

- $|f(x) - v(x)| \le \frac{h}{2} \max\limits_{a\le\xi\le b} |f^{'}(\xi)|$: Piecewise constant
- $|f(x) - v(x)| \le \frac{h^2}{8} \max\limits_{a\le\xi\le b} |f^{''}(\xi)|$: Piecewise linear
- $|f(x) - v(x)| \le \frac{h^4}{384} \max\limits_{a\le\xi\le b} |f^{''''}(\xi)|$: Piecewise cubic Hermite

Hermite cubic interpolant

Example: For the function $f(x) = \ln(x)$, we have the values $f(1) = 0, f^{'}(1) = 1, f(2) = 0.693147$, and $f^{'}(2) = 0.5$. Construct the corresponding Hermite cubic interpolant.

- $v(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$
- $v^{'}(x) = c_1 + 2c_2 x + 3c_3 x^2$
- $v(1) = f(1) = c_0 + c_1 + c_2 + c_3 = 0$
- $v^{'}(1) = f^{'}(1) = c_1 + 2c_2 + 3c_3 = 1$
- $v(2) = f(2) = c_0 + 2c_1 + 4c_2 + 8c_3 = 0.693147$
- $v^{'}(2) = f^{'}(2) = c_1 + 4c_2 + 12c_3 = \frac{1}{2}$
- $v(x) = -1.5343 + 2.1822x - 0.7617x^2 + 0.1137x^3$
- $f(x) = \ln(x)$, $[1, 2]$, $f^{''''}(\xi) = \frac{6}{\xi^4}$, $h = \frac{1}{4}$ (say). Interpolating error

$$|f(x) - v(x)| \leq \frac{6}{384} \left(\frac{1}{4}\right)^4 \approx 6 \times 10^{-5}$$

Cubic spline interpolation

Motivation: (Disadvantage of Herimite piecewise cubics)

- Need to evaluate $f^{'}(t_i)$, but f typically unknown!
- In some cases, C^1 may not provide sufficient smoothness

Given $\{(x_i, y_i)\}_{i=0}^{n} : a = x_0 < x_1 < \cdots < x - n - 1 < x_n = b$ and $y_i = f(x_i)$

- f: Unknown
- x_i: break points
- $4n$: parameters, $2n$: Interpolating conditions by a continuous interpolant, we use $2n$: conditions so that $v(x) \in C^2[a, b]$. The result is referred to as a **cubic spline**.
- $s_i(x_i) = f(x_i),\ i = 0, \cdots, n-1$
- $s_i(x_{i+1}) = f(x_{i+1}),\ i = 0, \cdots, n-1$
- $s_i^{'}(x_{i+1}) = s_{i+1}^{'}(x_{i+1}),\ i = 0, \cdots, n-2$
- $s_i^{''}(x_{i+1}) = s_{i+1}^{''}(x_{i+1}),\ i = 0, \cdots, n-2$
- 2 free conditions

Cubic spline interpolation

i	0	1	2
x_i	0.0	1.0	2.0
$f(x_i)$	1.1	0.9	2.0

- $s_0(x) = a_0 + b_0 x + c_0 x^2 + d_0 x^3$
- $s_1(x) = a_1 + b_1(x-1) + c_1(x-1)^2 + d_1(x-1)^3$
- $v(x) = \begin{cases} s_0(x) & x < 1 \\ s_1(x) & x \geq 1 \end{cases}$

Conditions

- $s_0(0) = f(0),\ s_0(1) = f(1)$
- $s_1(1) = f(1),\ s_1(2) = f(2)$
- $s_0^{'}(1) = s_1^{'}(1)$
- $s_0^{''}(1) = s_1^{''}(1)$

To additional conditions are needed:

Cubic spline interpolation

- **free boundary**, giving a **natural spline**
  - $v^{''}(x_0) = v^{''}(x_n) = 0$
    - cons, no prior reason to assume $f^{''} = 0$ at the endpoints
- **clamped boundary**, known as **complete spline**
  - $v^{'}(x_0) = f^{'}(x_0)$, $v^{'}(x_n) = f^{'}(x_n)$
    - cons, $f^{'}$ is available!
- **not-a-knot**
  - Enforce the continuity of the third derivative of the spline interpolant at the nearest interior break points, x_1 and x_{n-1}.