Numerical Analysis

Lecture 8: Interpolation & Polynomial Approximation

Instructor: Prof. Qiwei Zhan
Zhejiang University

Review

- Interpolation
- Taylor Polynomials
- Lagrange Interpolating Polynomials

Lagrange Interpolating Polynomial

Theorem: n-th Lagrange Interpolating Polynomial

If $x_0, x_1, \cdots, x_n$ are $n+1$ distinct numbers and f is a function whose values are given at these numbers, then a unique polynomial $P(x)$ of degree at most n exists with

$$f(x_k) = P(x_k), \quad \text{for each } k = 0, 1, \cdots, n.$$

The polynomial is given by

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

where, for each $k = 0, 1, \cdots, n$.

Lagrange Interpolating Polynomial

$$
\begin{aligned}
L_{n,k}(x) &= \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} \\
&= \prod_{i=0,i\neq k}^{n} \frac{x-x_i}{x_k-x_i}
\end{aligned}
$$

Lagrange Interpolating Polynomial

Theorem: Theoretical Error Bound

Suppose $x_0, x_1, \cdots, x_n$ are distinct numbers in the interval $[a, b]$ and $f \in C^{n+1}[a, b]$. Then, for each x in $[a, b]$, a number $\xi(x)$ (generally unknown) between $x_0, x_1, \cdots, x_n$, and hence in (a, b), exists with

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

where $P(x)$ is the interpolating polynomial given by

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

Lagrange Interpolating Polynomial

Remark

In general case, the theoretical Lagrange error term cannot be applied because we have no knowledge of the derivative of f.

Recursive Lagrange Polynomial Approximations

Theorem

Let f be defined at $x_0, x_1, \cdots, x_k$, and let x_j and x_i be two distinct numbers in this set. Then

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

is the kth Lagrange polynomial that interpolates f at the $k+1$ points $x_0, x_1, \cdots, x_k$.

Neville's Method

Neville's Iterated Interpolation Algorithm

```
INPUT   numbers $x, x_0, x_1, \ldots, x_n$; values $f(x_0), f(x_1), \ldots, f(x_n)$ as the first column
        $Q_{0,0}, Q_{1,0}, \ldots, Q_{n,0}$ of $Q$.

OUTPUT  the table $Q$ with $P(x) = Q_{n,n}$.

Step 1  For $i = 1, 2, \ldots, n$
            for $j = 1, 2, \ldots, i$
                set $Q_{i,j} = \dfrac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}}$.

Step 2  OUTPUT ($Q$);
        STOP.
```

- Neville's Algorithm generate successively higher-degree polynomial approximations at a specific point.

Recursive Lagrange Polynomial Approximations

Example

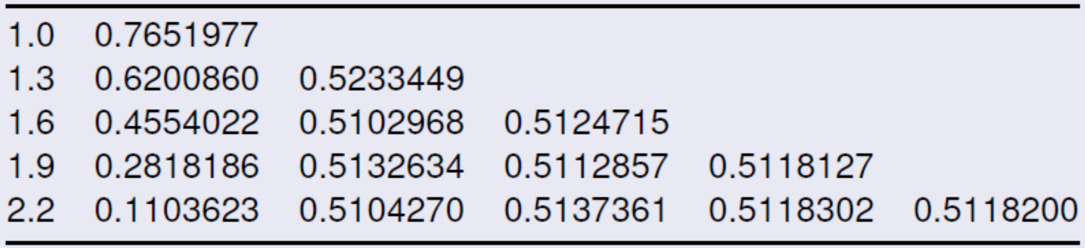
Values of various interpolating polynomials at $x = 1.5$ were obtained in an earlier example using the following data:

x	1.0	1.3	1.6	1.9	2.2
$f(x)$	0.7651977	0.6200860	0.4554022	0.2818186	0.1103623

Apply Neville's method to the data by constructing a recursive table in the Q-notation array format.

Recursive Lagrange Polynomial Approximations

Solution (6/6)



1.0	0.7651977				
1.3	0.6200860	0.5233449			
1.6	0.4554022	0.5102968	0.5124715		
1.9	0.2818186	0.5132634	0.5112857	0.5118127	
2.2	0.1103623	0.5104270	0.5137361	0.5118302	0.5118200

Outline

Newton’s Divided Difference Interpolating Polynomial

Question:

- How to successively generate the polynomials themselves?

Newton’s Divided Difference Interpolating Polynomial

Question:

- How to successively generate the polynomials themselves?

A New Algebraic Representation for $P_n(x)$

We first express $P_n(x)$ in the form

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

for appropriate constants $a_0, a_1, \cdots, a_n$.

Newton's Divided Difference Interpolating Polynomial

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

Newton's Divided Difference Interpolating Polynomial

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

a_0

$$a_0 = P_n(x_0) = f(x_0)$$

Newton's Divided Difference Interpolating Polynomial

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

a_0

$$a_0 = P_n(x_0) = f(x_0)$$

a_1

$$P_n(x_1) = f(x_0) + a_1(x_1 - x_0) = f(x_1) \quad \Rightarrow \quad a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

The Divided Difference Notation

Forward Difference Operator Δ

For a given sequence $\{p_n\}_{n=0}^{\infty}$, the forward difference Δp_n is defined by

$$\Delta p_n = p_{n+1} - p_n, \quad \text{for} \quad n \geq 0.$$

Higher powers of the operator Δ are defined recursively by

$$\Delta^k p_n = \Delta(\Delta^{k-1} p_n), \quad \text{for} \quad k \geq 2.$$

The Divided Difference Notation

The Divided Difference Notation

- The zeroth divided difference of the function f with respect to x_i, denoted $f[x_i]$, is simply the value of f at x_i
$$f[x_i] = f(x_i)$$
- The first divided difference of the function f with respect to x_i and x_{i+1} is denoted $f[x_i, x_{i+1}]$ and defined as
$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$
- The second divided difference, $f[x_i, x_{i+1}, x_{i+2}]$, is defined as
$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

The Divided Difference Notation

The Divided Difference Notation

- Similarly, after the $(k-1)$st divided differences,

$$f[x_i, x_{i+1}, x_{i+2}, \cdots, x_{i+k-1}] \text{ and } f[x_{i+1}, x_{i+2}, \cdots, x_{i+k-1}, x_{i+k}]$$

  have been determined, the kth divided difference relative to x_i, x_{i+1}, x_{i+2}, $\ldots$, x_{i+k} is

$$\begin{aligned} & f[x_i, x_{i+1}, \cdots, x_{i+k-1}, x_{i+k}] \\ & = \frac{f[x_{i+1}, x_{i+2}, \cdots, x_{i+k-1}, x_{i+k}] - f[x_i, x_{i+1}, x_{i+2}, \cdots, x_{i+k-1}]}{x_{i+k} - x_i} \end{aligned}$$

- The process ends with the single nth divided difference,

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

Newton's Divided Difference Interpolating Polynomial

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

Using the Divided Difference Notation

- Returning to the interpolating polynomial, we can now use the divided difference notation to write:

$$a_0 = f(x_0) = f[x_0]$$

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1]$$

- Hence, the interpolating polynomial is

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

Newton's Divided Difference Interpolating Polynomial

$$P_n(x) = f[x_0] + f[x_0,x_1](x-x_0) + a_2(x-x_0)(x-x_1) \\ + \cdots + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

$$\begin{aligned}
f(x_2) &= f(x_0) + f[x_0,x_1](x_2-x_0) + a_2(x_2-x_0)(x_2-x_1) \\
\Rightarrow a_2 &= \frac{f(x_2) - f(x_0) - f[x_0,x_1](x_2-x_0)}{(x_2-x_0)(x_2-x_1)} \\
\Rightarrow a_2 &= \frac{\dfrac{f(x_2)-f(x_1)}{x_2-x_1} - \dfrac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0} = f[x_0,x_1,x_2]
\end{aligned}$$

Newton's Divided Difference Interpolating Polynomial

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + a_2(x - x_0)(x - x_1)$$
$$+ \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

- As might be expected from the evaluation of a_0 and a_1, the required constants are

$$a_k = f[x_0, x_1, x_2, \cdots, x_k],$$

  for each $k = 0, 1, \cdots, n$.
- So $P_n(x)$ can be rewritten in a form called Newton's Divided-Difference:

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

Newton's Divided Difference Interpolating Polynomial

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

Newton's Divided Difference Interpolating Polynomial

Newton's Divided Difference Algorithm

INPUT numbers $x_0, x_1, \ldots, x_n$; values $f(x_0), f(x_1), \ldots, f(x_n)$ as $F_{0,0}, F_{1,0}, \ldots, F_{n,0}$.

OUTPUT the numbers $F_{0,0}, F_{1,1}, \ldots, F_{n,n}$ where

$$P_n(x) = F_{0,0} + \sum_{i=1}^{n} F_{i,i} \prod_{j=0}^{i-1} (x - x_j). \quad (F_{i,i} \text{ is } f[x_0, x_1, \ldots, x_i].)$$

```
Step 1  For $i = 1, 2, \ldots, n$
            For $j = 1, 2, \ldots, i$
                set $F_{i,j} = \dfrac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$.   $(F_{i,j} = f[x_{i-j}, \ldots, x_i].)$
Step 2  OUTPUT $(F_{0,0}, F_{1,1}, \ldots, F_{n,n})$;
        STOP.
```

Remarks

Remarks

- High-degree polynomials, such as Lagrange polynomials, can oscillate erratically. That is, a minor fluctuation over a small portion of the interval can induce large fluctuations over the entire range.

Remarks

Remarks

- High-degree polynomials, such as Lagrange polynomials, can oscillate erratically. That is, a minor fluctuation over a small portion of the interval can induce large fluctuations over the entire range.
- An alternative approach is to divide the approximation interval into a collection of subintervals and construct a different approximating polynomial on each subinterval. This is called piecewise-polynomial approximation.

Outline

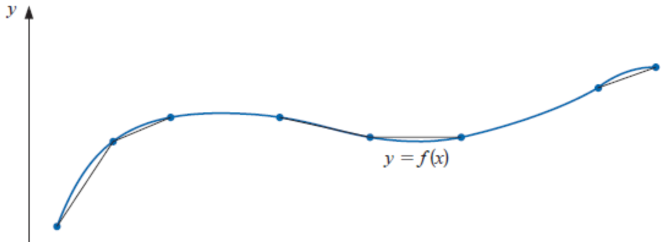
Piecewise-Polynomial Approximation

Piecewise-linear Interpolation

The simplest piecewise-polynomial approximation is piecewise-linear interpolation, which consists of joining a set of data points

$$\{(x_0, f(x_0)), (x_1, f(x_1)), \cdots, (x_n, f(x_n))\}$$

by a series of straight lines:



Piecewise-Polynomial Approximation

Disadvantage of Piecewise-linear Interpolation

- There is likely no differentiability at the endpoints of the subintervals, which, in a geometrical context, means that the interpolating function is not “smooth”.
- Often it is clear from physical conditions that smoothness is required, so the approximating function must be continuously differentiable.

Piecewise-Polynomial Approximation

Differentiable Piecewise-polynomial Function

- The simplest type of differentiable piecewise-polynomial function on an entire interval $[x_0, x_n]$ is the function obtained by fitting one quadratic polynomial between each successive pair of nodes.
- This is done by constructing a quadratic on

$$[x_0, x_1] \text{ agreeing with the function at } x_0 \text{ and } x_1,$$

and another quadratic on

$$[x_1, x_2] \text{ agreeing with the function at } x_1 \text{ and } x_2,$$

and so on.

Piecewise-Polynomial Approximation

Differentiable Piecewise-polynomial Function (cont'd)

- A general quadratic polynomial has three arbitrary constants, the constant term, the coefficient of x, and the coefficient of x^2, and only two conditions are required to fit the data at the endpoints of each subinterval.
- So flexibility exists that permits the quadratics to be chosen so that the interpolant has a continuous derivative on $[x_0, x_n]$.
- The difficulty arises because we generally need to specify conditions about the derivative of the interpolant at the endpoints x_0 and x_n.
- There is not a sufficient number of constants to ensure that the conditions will be satisfied.

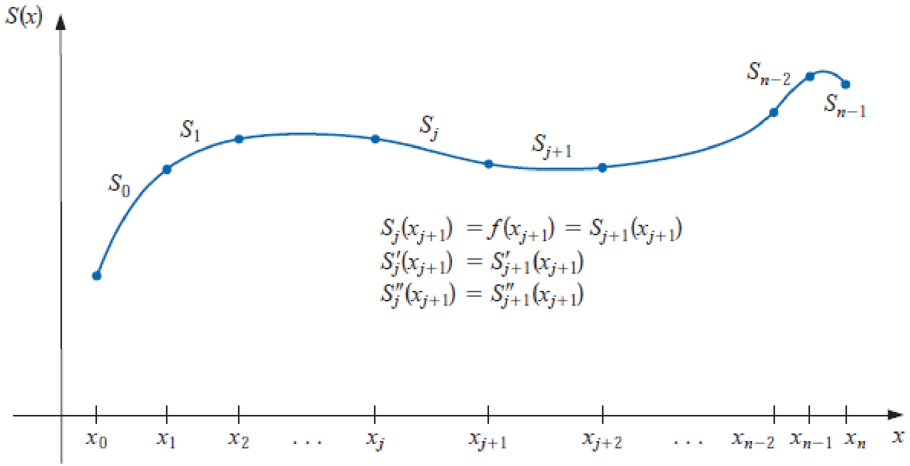
Cubic Splines: Establishing Conditions

Most Common Piecewise-polynomial Approximation

- The most common piecewise-polynomial approximation uses cubic polynomials between each successive pair of nodes and is called cubic spline interpolation.
- A general cubic polynomial involves four constants, so there is sufficient flexibility in the cubic spline procedure to ensure that the interpolant is not only continuously differentiable on the interval, but also has a continuous second derivative.

Cubic Splines: Establishing Conditions

The construction of the cubic spline does not, however, assume that the derivatives of the interpolant agree with those of the function it is approximating, even at the nodes.



$S(x)$
S_0
S_1
S_j
S_{j+1}
S_{n-2}
S_{n-1}
$S_j(x_{j+1}) = f(x_{j+1}) = S_{j+1}(x_{j+1})$
$S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$
$S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$
x_0
x_1
x_2
...
x_j
x_{j+1}
x_{j+2}
...
x_{n-2}
x_{n-1}
x_n
x

Cubic Spline Interpolant

Definition

Given a function f defined on $[a, b]$ and a set of nodes $a = x_0 < x_1 < \cdots < x_n = b$, a cubic spline interpolant S for f is a function that satisfies the following conditions:

1. $S(x)$ is a cubic polynomial, denoted $S_j(x)$, on the subinterval $[x_j, x_{j+1}]$ for each $j = 0, 1, \cdots, n-1$;
2. $S_j(x_j) = f(x_j)$ and $S_j(x_{j+1}) = f(x_{j+1})$ for each $j = 0, 1, \cdots, n-1$;
3. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ for each $j = 0, 1, \cdots, n-2$;
4. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ for each $j = 0, 1, \cdots, n-2$;
5. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ for each $j = 0, 1, \cdots, n-2$;
6. One of the following sets of boundary conditions is satisfied:
    - $S''(x_0) = S''(x_n) = 0$ (natural (or free) boundary);
    - $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (clamped boundary).

Cubic Spline Interpolant

Natural & Clamped Boundary Conditions

- Although cubic splines are defined with other boundary conditions, the conditions given in (6) are sufficient for our purposes.
- When the free boundary conditions occur, the spline is called as a natural spline, and its graph approximates the shape that a long flexible rod would assume if forced to go through the data points $(x_0, f(x_0)), (x_1, f(x_1)), \cdots, (x_n, f(x_n))$.
- In general, clamped boundary conditions lead to more accurate approximations because they include more information about the function.
- However, for this type of boundary condition to hold, it is necessary to have either the values of the derivative at the endpoints or an accurate approximation to those values.

Cubic Spline Interpolant

Example: 3 Data Values

Construct a natural cubic spline that passes through the points (1, 2), (2, 3) and (3, 5).

Cubic Spline Interpolant

Example: 3 Data Values

Construct a natural cubic spline that passes through the points (1, 2), (2, 3) and (3, 5).

Solution (1/3)

This spline consists of two cubics: the first for the interval $[1, 2]$, denoted

$$S_0(x) = a_0 + b_0(x-1) + c_0(x-1)^2 + d_0(x-1)^3,$$

and the other for $[2, 3]$, denoted

$$S_1(x) = a_1 + b_1(x-2) + c_1(x-2)^2 + d_1(x-2)^3.$$

Cubic Spline Interpolant

Solution (2/3)

There are 8 constants to be determined, which requires 8 conditions, 4 conditions come from the fact that the splines must agree with the data at the nodes. Hence

$$2 = f(1) = a_0, \;\; 3 = f(2) = a_0 + b_0 + c_0 + d_0;$$
$$3 = f(2) = a_1, \;\; 5 = f(3) = a_1 + b_1 + c_1 + d_1$$

2 more come from the fact that $S_0'(2) = S_1'(2)$ and $S_0''(2) = S_1''(2)$. These are

$$S_0'(2) = S_1'(2): \quad b_0 + 2c_0 + 3d_0 = b_1$$
$$S_0''(2) = S_1''(2): \quad 2c_0 + 6d_0 = 2c_1$$

Cubic Spline Interpolant

Solution (3/3)

The final 2 come from the natural boundary conditions:

$$\begin{aligned} S_0''(1) = 0: &\quad 2c_0 = 0 \\ S_1''(3) = 0: &\quad 2c_1 + 6d_1 = 0. \end{aligned}$$

Solving this system of 8 equations gives the spline:

$$S(x) = \begin{cases} 2 + \frac{3}{4}(x-1) + \frac{1}{4}(x-1)^3, & \text{for } x \in [1,2] \\ 3 + \frac{3}{2}(x-2) + \frac{3}{4}(x-2)^2 - \frac{1}{4}(x-2)^3, & \text{for } x \in [2,3] \end{cases}$$

Outline

Cubic Spline Interpolant

Basic Approach

- A spline defined on an interval that is divided into n subintervals will require determining $4n$ constants.
- To construct the cubic spline interpolant for a given function f, the conditions in the definition are applied to the cubic polynomials

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

for each $j = 0, 1, \cdots, n-1$. Since $S_j(x_j) = a_j = f(x_j)$, Condition (3) can be applied to obtain

$$\begin{aligned} a_{j+1} &= S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \\ &= a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3, \end{aligned}$$

for each $j = 0, 1, \cdots, n-2$.

Cubic Spline Interpolant

$$a_{j+1} = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$$

Basic Approach (cont'd)

The terms $x_{j+1} - x_j$ are used repeatedly in this development, so it is convenient to introduce the simpler notation

$$h_j = x_{j+1} - x_j,$$

for each $j = 0, 1, \cdots, n-1$.
If we also define $a_n := f(x_n)$, then the equation

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$$

holds for each $j = 0, 1, \cdots, n-1$.

Cubic Spline Interpolant

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$$

Basic Approach (cont'd)

In a similar manner, define $b_n := S'(x_n)$ and observe that

$$S_j'(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2$$

implies $S_j'(x_j) = b_j$, for each $j = 0, 1, \cdots, n-1$. Applying Condition (4) gives

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2,$$

for each $j = 0, 1, \cdots, n-1$.

Cubic Spline Interpolant

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$$
$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2$$

Basic Approach (cont'd)

Another relationship between the coefficients of S_j is obtained by defining $c_n := S''(x_n)/2$ and applying condition (5), namely $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$. Then, for each $j = 0, 1, \cdots, n-1$,

$$c_{j+1} = c_j + 3d_j h_j$$

Cubic Spline Interpolant

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$$

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2$$

$$c_{j+1} = c_j + 3d_j h_j$$

Basic Approach (cont'd)

Solving for d_j in the third equation and substituting this value into the other two gives, for each $j = 0, 1, \cdots, n-1$, the new equations

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1})$$

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

Cubic Spline Interpolant

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1})$$
$$b_{j+1} = b_j + h_j(c_j + c_{j+1})$$

Basic Approach (cont'd)

The final relationship involving the coefficients is obtained by solving the appropriate equation in the form of equation a_{j+1} above, first for b_j:

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

and then, with a reduction of the index, for b_{j-1}:

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j)$$

Cubic Spline Interpolant

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j)$$

Basic Approach (cont'd)

Substituting these values into the equation derived from

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

with the index reduced by one, gives the linear system of equations

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_jc_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}),$$

Cubic Spline Interpolant

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

Basic Approach (cont'd)

- This system involves only the $\{c_j\}_{j=0}^{n}$ as unknowns.
- The values of $\{h_j\}_{j=0}^{n-1}$ and $\{a_j\}_{j=0}^{n}$ are given, respectively, by the spacing of the nodes $\{x_j\}_{j=0}^{n}$ and the values of f at the nodes.
- So once the values of $\{c_j\}_{j=0}^{n}$ are determined, it is a simple matter to find the remainder of the constants $\{b_j\}_{j=0}^{n-1}$ and $\{d_j\}_{j=0}^{n-1}$. Then we can construct the cubic polynomials $\{S_j(x)\}_{j=0}^{n-1}$.

Cubic Spline Interpolant

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

Basic Approach (cont'd)

- The major question that arises in connection with this construction is whether the values of $\{c_j\}_{j=0}^{n}$ can be found using the system of equations given above and, if so, whether these values are unique.
- We will answer this question using theorems which indicate that this is the case when either of the boundary conditions given in part (6) of the definition are imposed.

Existence of A Unique Natural Spline Interpolant

Theorem

If f is defined at $a = x_0 < x_1 < \cdots < x_n = b$, then f has a unique natural spline interpolant S on the nodes $x_0, x_1, \cdots, x_n$; that is, a spline interpolant that satisfies the natural boundary conditions $S''(a) = 0$ and $S''(b) = 0$.

Existence of A Unique Natural Spline Interpolant

Proof (1/4)

- Using the notation

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

  the boundary conditions in this case imply that $c_n = \frac{1}{2}S''_n(x_n) = 0$ and that $0 = S''(x_0) = 2c_0 + 6d_0(x_0 - x_0)$ so $c_0 = 0$.
- The two equations $c_0 = 0$ and $c_n = 0$ together with the equations

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_jc_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

  produce a linear system described by the vector equation $A\mathbf{x} = \mathbf{b}$.

Existence of A Unique Natural Spline Interpolant

Proof (2/4)

A is the $(n+1) \times (n+1)$ matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & \ddots & & \vdots \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix}$$

Existence of A Unique Natural Spline Interpolant

Proof (3/4)

b and **x** are the vectors

$$\mathbf{b} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Existence of A Unique Natural Spline Interpolant

Proof (4/4)

- The matrix A is strictly diagonally dominant, that is, in each row the magnitude of the diagonal entry exceeds the sum of the magnitudes of all the other entries in the row.
- A linear system with a matrix of this form will be shown to have a unique solution for $c_0, c_1, \cdots, c_n$.

Natural Cubic Spline Algorithm

To construct the cubic spline interpolant S for the function f, defined at the numbers $x_0 < x_1 < \cdots < x_n$, satisfying $S''(x_0) = S''(x_n) = 0$:

INPUT $n; x_0, x_1, \ldots, x_n; a_0 = f(x_0), a_1 = f(x_1), \ldots, a_n = f(x_n)$.

OUTPUT a_j, b_j, c_j, d_j for $j = 0, 1, \ldots, n-1$.

(*Note:* $S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$ *for* $x_j \le x \le x_{j+1}$.)

Step 1 For $i = 0, 1, \ldots, n-1$ set $h_i = x_{i+1} - x_i$.

Step 2 For $i = 1, 2, \ldots, n-1$ set

$$\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1}).$$

Step 3 Set $l_0 = 1$; (*Steps 3, 4, 5, and part of Step 6 solve a tridiagonal linear system using a method described in Algorithm 6.7.*)

$\mu_0 = 0$;
$z_0 = 0$.

Natural Cubic Spline Algorithm (cont'd)

```
Step 4  For $i = 1, 2, \ldots, n-1$
            set $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$;
                $\mu_i = h_i / l_i$;
                $z_i = (\alpha_i - h_{i-1} z_{i-1}) / l_i$.

Step 5  Set $l_n = 1$;
            $z_n = 0$;
            $c_n = 0$.

Step 6  For $j = n-1, n-2, \ldots, 0$
            set $c_j = z_j - \mu_j c_{j+1}$;
                $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$;
                $d_j = (c_{j+1} - c_j)/(3h_j)$.

Step 7  OUTPUT ($a_j, b_j, c_j, d_j$ for $j = 0, 1, \ldots, n-1$);
        STOP.
```

■

Natural Cubic Spline Interpolant

Example: $f(x) = e^x$

Use the data points (0, 1), (1, e), $(2, e^2)$ and $(3, e^3)$ to form a natural spline $S(x)$ that approximates $f(x) = e^x$.

Natural Cubic Spline Interpolant

Example: $f(x) = e^x$

Use the data points (0, 1), (1, e), $(2, e^2)$ and $(3, e^3)$ to form a natural spline $S(x)$ that approximates $f(x) = e^x$.

Solution (1/7)

With $n = 3$, $h_0 = h_1 = h_2 = 1$ and the notation

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

for $x_j \leq x \leq x_{j+1}$, we have

- $a_0 = 1, a_1 = e$
- $a_2 = e^2, a_3 = e^3$

Natural Cubic Spline Interpolant

Solution (2/7)

So the matrix **A** and the vectors **b** and **x** given in the Natural Spline Theorem have the forms

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 3(e^2 - 2e + 1) \\ 3(e^3 - 2e^2 + e) \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

The vector-matrix equation $\mathbf{Ax} = \mathbf{b}$ is equivalent to the system:

$$\begin{aligned} c_0 &= 0 \\ c_0 + 4c_1 + c_2 &= 3(e^2 - 2e + 1) \\ c_1 + 4c_2 + c_3 &= 3(e^3 - 2e^2 + e) \\ c_3 &= 0 \end{aligned}$$

Natural Cubic Spline Interpolant

Solution (3/7)

This system has the solution $c_0 = c_3 = 0$ and, to 5 decimal places,

$$\begin{aligned} c_1 &= \frac{1}{5}(-e^3 + 6e^2 - 9e + 4) \approx 0.75685 \\ c_2 &= \frac{1}{5}(4e^3 - 9e^2 + 6e - 1) \approx 5.83007 \end{aligned}$$

Natural Cubic Spline Interpolant

Solution (4/7)

Solving for the remaining constants gives

$$\begin{aligned}
b_0 &= \frac{1}{h_0}(a_1 - a_0) - \frac{h_0}{3}(c_1 + 2c_0) \\
&= (e-1) - \frac{1}{15}(-e^3 + 6e^2 - 9e + 4) \approx 1.46600 \\
b_1 &= \frac{1}{h_1}(a_2 - a_1) - \frac{h_1}{3}(c_2 + 2c_1) \\
&= (e^2 - e) - \frac{1}{15}(2e^3 + 3e^2 - 12e + 7) \approx 2.22285 \\
b_2 &= \frac{1}{h_2}(a_3 - a_2) - \frac{h_2}{3}(c_3 + 2c_2) \\
&= (e^3 - e^2) - \frac{1}{15}(8e^3 - 18e^2 + 12e - 2) \approx 8.80977
\end{aligned}$$

Natural Cubic Spline Interpolant

Solution (5/7)

$$\begin{aligned}
d_0 &= \frac{1}{3h_0}(c_1 - c_0) = \frac{1}{15}(-e^3 + 6e^2 - 9e + 4) \approx 0.25228 \\
d_1 &= \frac{1}{3h_1}(c_2 - c_1) = \frac{1}{3}(e^3 - 3e^2 + 3e - 1) \approx 1.69107 \\
d_2 &= \frac{1}{3h_2}(c_3 - c_2) = \frac{1}{15}(-4e^3 + 9e^2 - 6e + 1) \approx -1.94336
\end{aligned}$$

Natural Cubic Spline Interpolant

Solution (6/7)

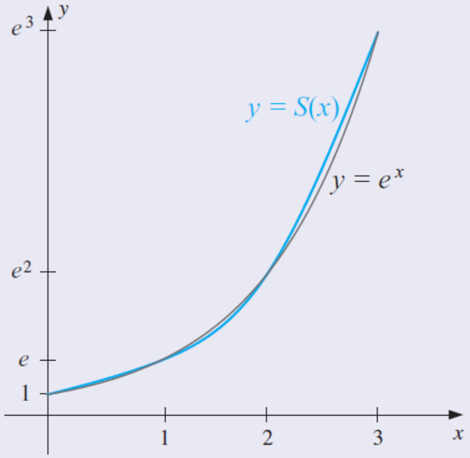
The natural cubic spine is described piecewisely by

$$S(x) = \begin{cases} 1 + 1.46600x + 0.25228x^3, & \text{for } x \in [0, 1] \\ 2.71828 + 2.22285(x-1) + 0.75685(x-1)^2 + 1.69107(x-1)^3, & \text{for } x \in [1, 2] \\ 7.38906 + 8.80977(x-2) + 5.83007(x-2)^2 - 1.94336(x-2)^3, & \text{for } x \in [2, 3] \end{cases}$$

The spline and its agreement with $f(x) = e^x$ are as shown in the following diagram.

Natural Cubic Spline Interpolant

Solution (7/7)



Unique Clamped Spline Interpolant

Theorem

If f is defined at $a = x_0 < x_1 < \cdots < x_n = b$ and differentiable at a and b, then f has a unique clamped spline interpolant S on the nodes $x_0, x_1, \cdots, x_n$; that is, a spline interpolant that satisfies the clamped boundary conditions $S'(a) = f'(a)$ and $S'(b) = f'(b)$.

Note: With $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$, for each $j = 0, 1, \cdots, n-1$, the proof will rely on two equations established earlier, namely:

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

Unique Clamped Spline Interpolant

$$(A) \qquad b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

Proof(1/5)

Since $f'(a) = S'(a) = S'(x_0) = b_0$, eq. (A) with $j = 0$ implies

$$f'(a) = \frac{1}{h_0}(a_1 - a_0) - \frac{h_0}{3}(2c_0 + c_1)$$

Consequently,

$$2h_0c_0 + h_0c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a)$$

Unique Clamped Spline Interpolant

$$(A) \qquad b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

Proof(2/5)

Similarly,

$$f'(b) = b_n = b_{n-1} + h_{n-1}(c_{n-1} + c_n)$$

so eq. (A) with $j = n - 1$ implies that

$$\begin{aligned} f'(b) &= \frac{a_n - a_{n-1}}{h_{n-1}} - \frac{h_{n-1}}{3}(2c_{n-1} + c_n) + h_{n-1}(c_{n-1} + c_n) \\ &= \frac{a_n - a_{n-1}}{h_{n-1}} + \frac{h_{n-1}}{3}(c_{n-1} + 2c_n) \end{aligned}$$

and $\quad h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3f'(b) - \dfrac{3}{h_{n-1}}(a_n - a_{n-1})$

Unique Clamped Spline Interpolant

Proof(3/5)

The equations

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

together with

$$2h_0c_0 + h_0c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a)$$

and

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1})$$

determine the linear system $\mathbf{Ax} = \mathbf{b}$

Unique Clamped Spline Interpolant

Proof(4/5)

$$\mathbf{A} = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

Unique Clamped Spline Interpolant

Proof(5/5)

$$\mathbf{b} = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix} \quad \text{with} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Matrix A is strictly diagonally dominant and a linear system with a matrix of this form can be shown to have a unique solution for

$c_0, c_1, \cdots, c_n$.

Clamped Cubic Spline Algorithm

Context

To construct the cubic spline interpolant S for the function f defined at the numbers $x_0 < x_1 < \cdots < x_n$, satisfying

$$S'(x_0) = f'(x_0) \quad \text{and} \quad S'(x_n) = f'(x_n)$$

where

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

for $x_j \leq x \leq x_{j+1}$.

Clamped Cubic Spline Algorithm (cont'd)

```
INPUT    $n;\ x_0, x_1, \ldots, x_n;\ a_0 = f(x_0),\ a_1 = f(x_1), \ldots, a_n = f(x_n);$
         $FPO = f'(x_0);\ FPN = f'(x_n).$
OUTPUT   $a_j, b_j, c_j, d_j$ for $j = 0, 1, \ldots, n-1$

Step 1   For $i = 0, 1, \ldots, n-1$ set $h_i = x_{i+1} - x_i$
Step 2   Set $\alpha_0 = 3(a_1 - a_0)/h_0 - 3FPO$
           $\alpha_n = 3FPN - 3(a_n - a_{n-1})/h_{n-1}$
Step 3   For $i = 1, 2, \ldots, n-1$
           set $\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$
```

(Note: In what follows, Steps 4, 5, 6 and part of Step 7 solve a tridiagonal linear system using a Crout Factorization algorithm.)

Clamped Cubic Spline Algorithm

```
Step 4  Set $l_0 = 2h_0$
            $\mu_0 = 0.5$
            $z_0 = \alpha_0 / l_0$
Step 5  For $i = 1, 2, \ldots, n-1$
            set $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$
                $\mu_i = h_i / l_i$
                $z_i = (\alpha_i - h_{i-1} z_{i-1}) / l_i$
Step 6  Set $l_n = h_{n-1}(2 - \mu_{n-1})$
            $z_n = (\alpha_n - h_{n-1} z_{n-1}) / l_n$
            $c_n = z_n$
Step 7  For $j = n-1, n-2, \ldots, 0$
            set $c_j = z_j - \mu_j c_{j+1}$
                $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$
                $d_j = (c_{j+1} - c_j)/(3h_j)$
Step 8  OUTPUT $(a_j, b_j, c_j, d_j$ for $j = 0, 1, \ldots, n-1)$ & STOP
```

Clamped Cubic Spline Interpolant

Example: $f(x) = e^x$

Use the data points (0, 1), (1, e), $(2, e^2)$ and $(3, e^3)$ to form a clamped spline $S(x)$ that approximates $f(x) = e^x$, with the additional information that $f'(0) = 1$ and $f'(3) = e^3$.

Clamped Cubic Spline Interpolant

Solution(1/5)

As before, we have $n = 3$, $h_0 = h_1 = h_2 = 1$, $a_0 = 1, a_1 = e$, $a_2 = e^2, a_3 = e^3$. This together with the information that $f'(0) = 1$ and $f'(3) = e^3$ gives the matrix **A** and the vectors **b** and **x** with the forms:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 3(e-2) \\ 3(e^2 - 2e + 1) \\ 3(e^3 - 2e^2 + e) \\ 3e^2 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Clamped Cubic Spline Interpolant

Solution(2/5)

The vector-matrix equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ is equivalent to the system of equations

$$\begin{aligned} 2c_0 + c_1 &= 3(e-2) \\ c_0 + 4c_1 + c_2 &= 3(e^2 - 2e + 1) \\ c_1 + 4c_2 + c_3 &= 3(e^3 - 2e^2 + e) \\ c_2 + 2c_3 &= 3e^2 \end{aligned}$$

Clamped Cubic Spline Interpolant

Solution(3/5)

This system has the solution to 5 decimal places,

$$\begin{aligned}
c_0 &= \frac{1}{15}(2e^3 - 12e^2 + 42e - 59) \approx 0.44468 \\
c_1 &= \frac{1}{15}(-4e^3 + 24e^2 - 39e + 28) \approx 1.26548 \\
c_2 &= \frac{1}{15}(14e^3 - 39e^2 + 24e - 8) \approx 3.35087 \\
c_3 &= \frac{1}{15}(-7e^3 + 42e^2 - 12e + 4) \approx 9.40815
\end{aligned}$$

Clamped Cubic Spline Interpolant

```
Step 7    For $j = n-1, n-2, \cdots, 0$
              set $c_j = z_j - \mu_j c_{j+1}$
                  $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$
                  $d_j = (c_{j+1} - c_j)/(3h_j)$
```

Solution(4/5)

Solving for the remaining constants (using Step 7 of the algorithm) gives

$$b_0 = 1.00000, \qquad b_1 = 2.71016, \qquad b_2 = 7.32652$$

and

$$d_0 = 0.27360, \qquad d_1 = 0.69513, \qquad d_2 = 2.01909$$

Clamped Cubic Spline Interpolant

Solution(5/5)

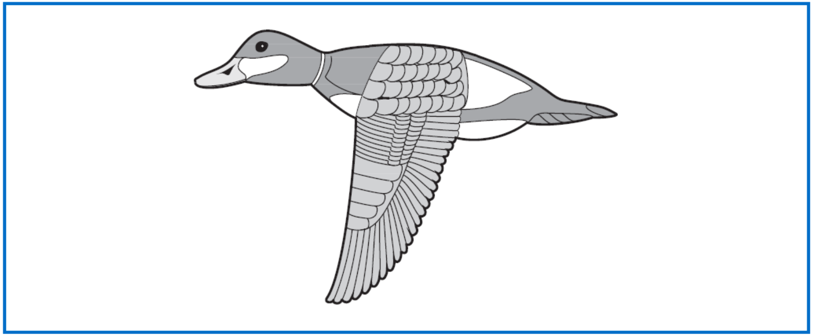
This gives the clamped cubic spline

$$S(x) = \begin{cases} 1 + x + 0.44468x^2 + 0.27360x^3, & \text{for } x \in [0, 1] \\ 2.71828 + 2.71016(x-1) + 1.26548(x-1)^2 + 0.69513(x-1)^3, & \text{for } x \in [1, 2] \\ 7.38906 + 7.32652(x-2) + 3.35087(x-2)^2 + 2.01909(x-2)^3, & \text{for } x \in [2, 3] \end{cases}$$

The graph of the clamped spline and $f(x) = e^x$ are so similar that no difference can be seen.

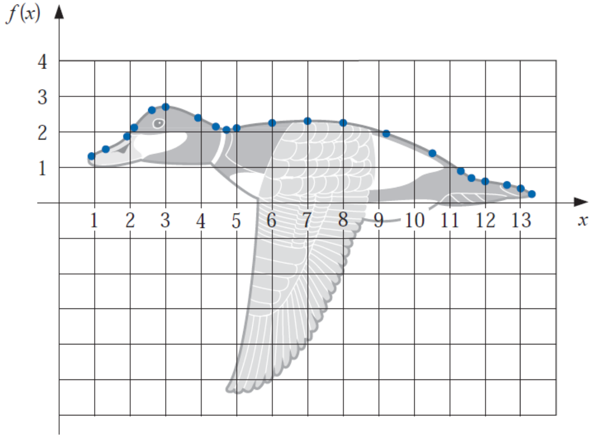
Real Model

Figure 3.11



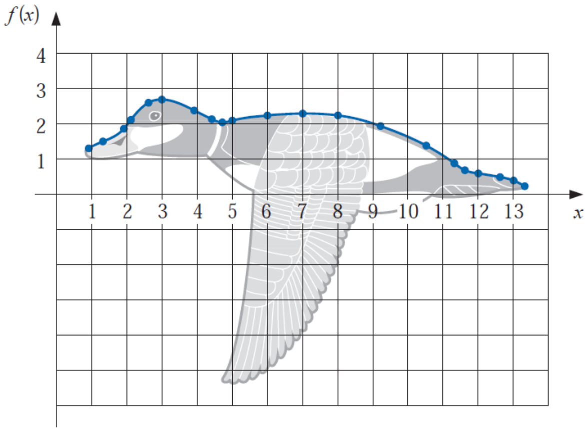
Real Model with Grids

Figure 3.12



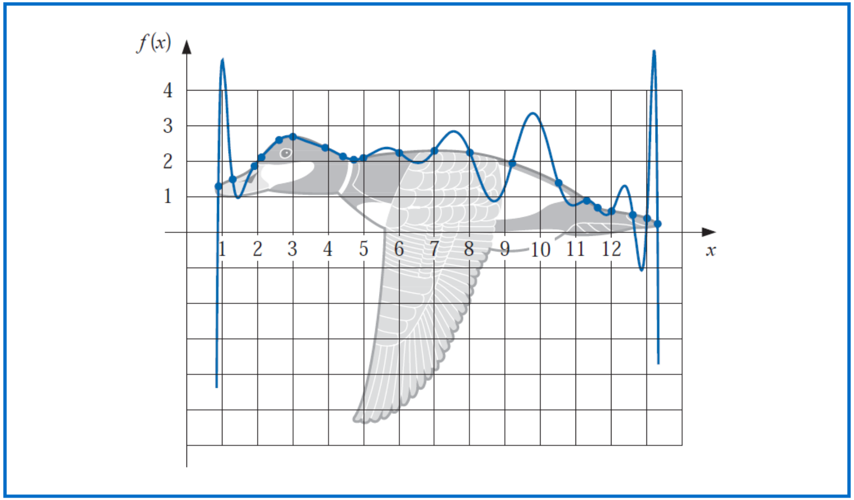
Real Model with the Natural Spline Interpolation

Figure 3.13



Real Model with the Lagrange Interpolation

Figure 3.14



Assignment

- Reading Assignment: Chap 3
- Homework: Assignment 4