MATH 4513 Numerical Analysis

Chapter 3. Interpolation and Polynomial Approximation

Xu Zhang

Department of Mathematics
Oklahoma State University

Text Book: Numerical Analysis (10th edition)
R. L. Burden, D. J. Faires, A. M. Burden

Table of Contents

3.1 Interpolation and Lagrange Polynomials

- One of the most useful classes of functions is **polynomials**:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0.$$

  where n is a nonnegative integer and $a_0, a_1, \cdots, a_n$ are real constants.

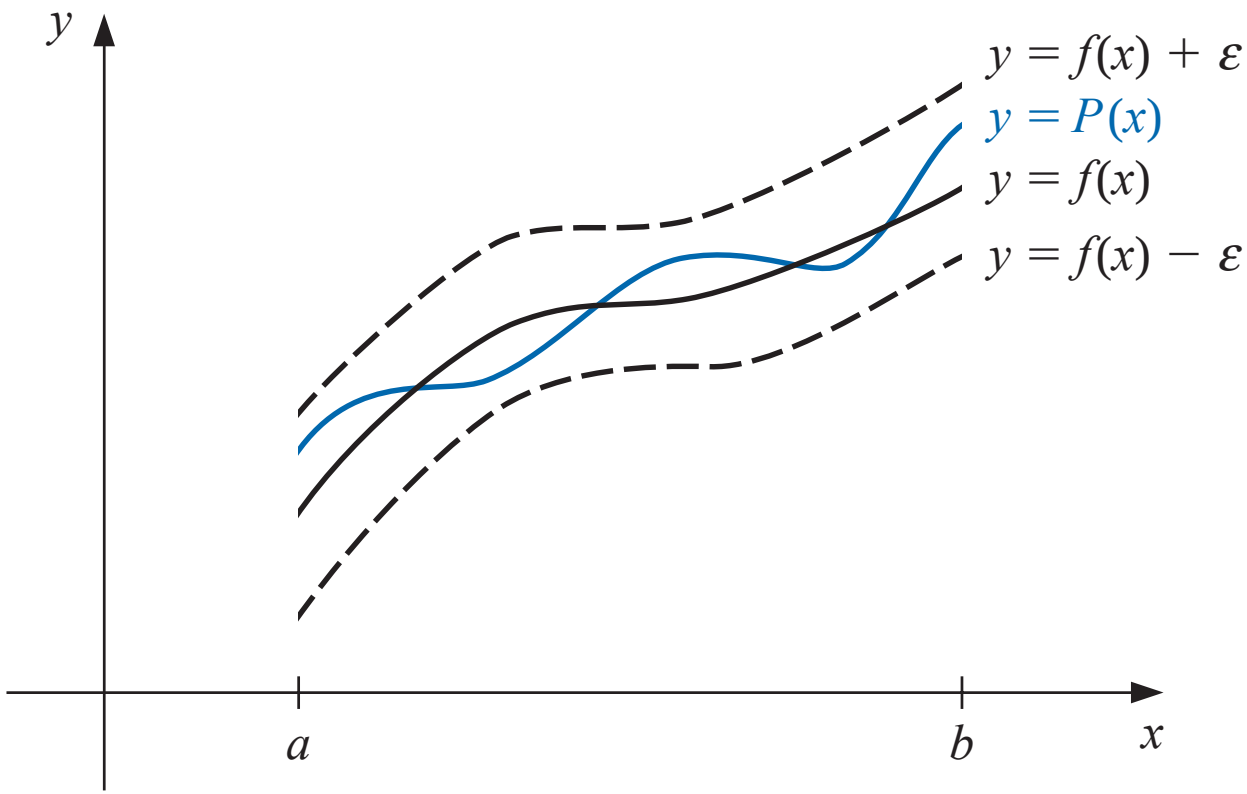
- One reason is that any continuous function can be approximated by a polynomial arbitrarily close. By this we mean that given any continuous function, there exists a polynomial that is as "close" to the given function as desired. (see Theorem 1)

- Another important reason for considering the class of polynomials in the approximation of functions is that the derivative and indefinite integral of a polynomial are easy to determine and are also polynomials.

Theorem 1 (Weierstrass Approximation Theorem).

Suppose $f \in C[a, b]$. *For each* $\epsilon > 0$, *there exists a polynomial* $P(x)$ *such that*

$$|f(x) - P(x)| < \epsilon, \quad \text{for all } x \in [a, b].$$

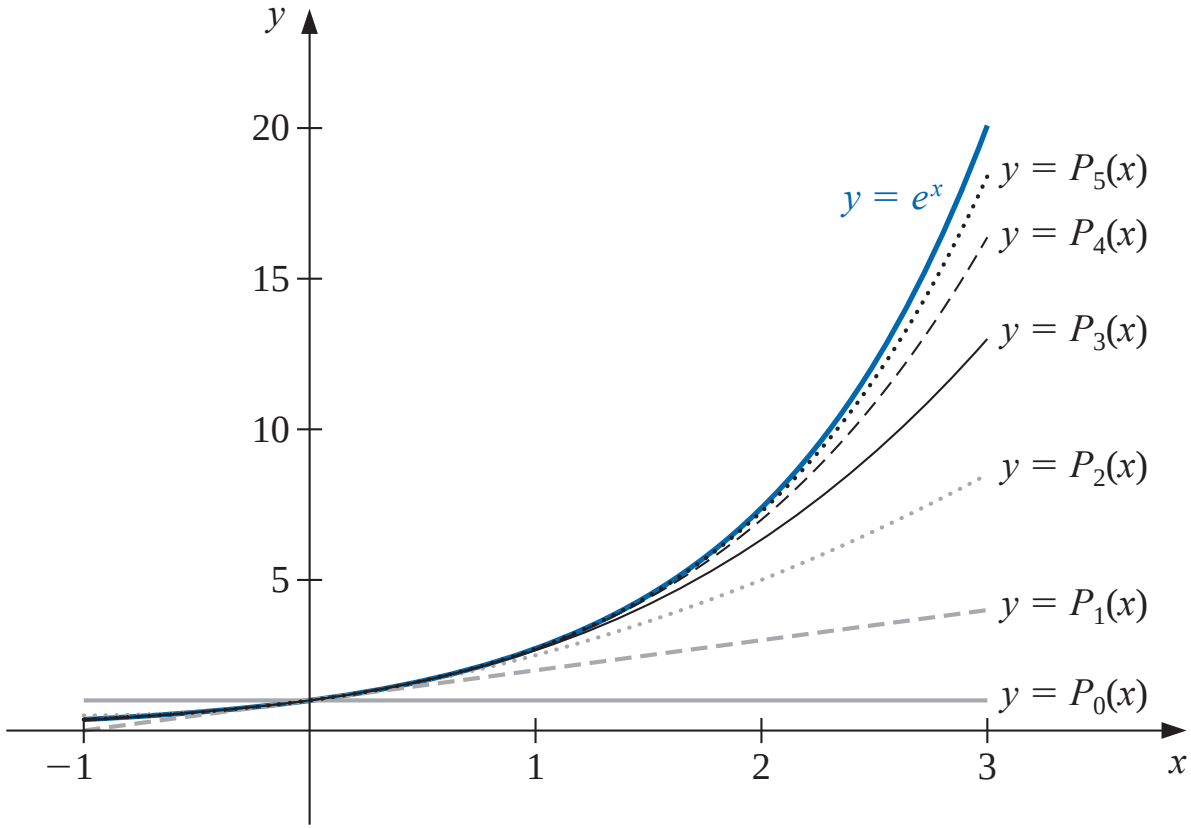


- A typical example for polynomial approximations is the Taylor polynomial.

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

- For example, using Taylor polynomials at $x_0 = 0$ to approximate $f(x) = e^x$, we obtain

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



Note that even for higher-degree polynomials, error becomes progressively worse as we move away from the point $x_0 = 0$.

$$f(3) = 20.0855$$

$$P_1(3) = 4, \quad P_2(3) = 8.5, \quad P_3(3) = 13, \quad P_4(3) = 16.375.$$

Remark on Taylor Polynomials

- For Taylor polynomials, all information used in the approximation is concentrated at the single number x_0, so these polynomials will generally give inaccurate approximations as we move away from x_0. This limits Taylor polynomial approximation to the situation in which approximations are needed only at numbers close to x_0.
- A good interpolation polynomial needs to provide a relatively accurate approximation over an entire interval, and Taylor polynomials do not generally do this.
- It is usually more efficient to develop methods that use **information spreaded at various points**.

Lagrange Interpolation Polynomials

- Suppose that a function $f(x)$ passes through two points (x_0, y_0) and (x_1, y_1). Define the following **linear Lagrange polynomials**

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad L_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

- It is easy to verify that

$$L_0(x_0) = 1, \quad L_0(x_1) = 0,$$

$$L_1(x_0) = 0, \quad L_1(x_1) = 1.$$

- The **linear Lagrange interpolating polynomial** through (x_0, y_0) and (x_1, y_1) is

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

- It can be verified that $P(x_0) = y_0$ and $P(x_1) = y_1$. **(Exercise)**

Example 2.

Determine the linear Lagrange interpolating polynomial that passes through the points $(2, 4)$ and $(5, 1)$.

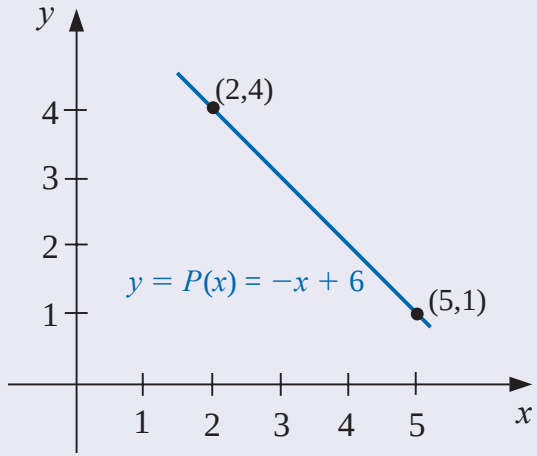
Solution

- In this case, we have

$$L_0(x) = \frac{x-5}{2-5} = -\frac{1}{3}(x-5), \quad \text{and} \quad L_1(x) = \frac{x-2}{5-2} = \frac{1}{3}(x-2).$$

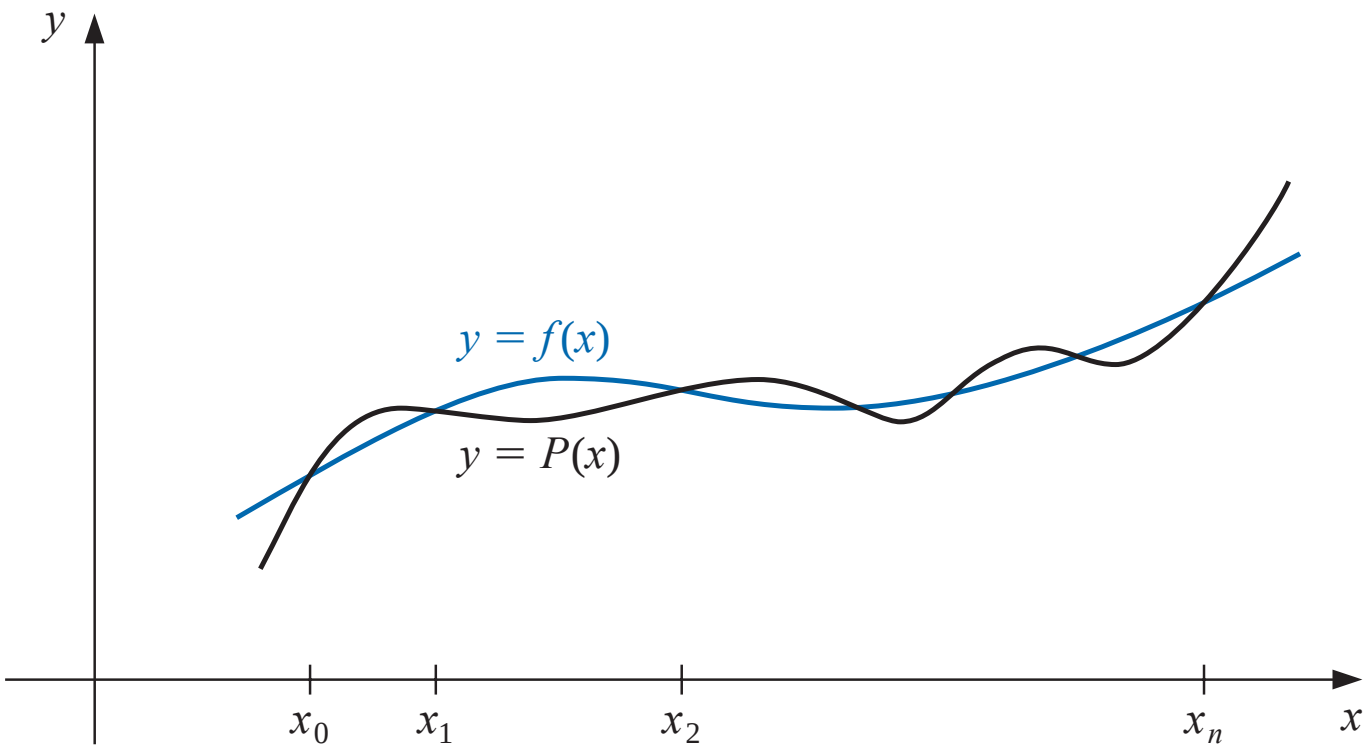
- Then, the linear Lagrange interpolating polynomial is

$$\begin{aligned} P(x) &= y_0 L_0(x) + y_1 L_1(x) \\ &= -\frac{1}{3}(x-5)\cdot 4 + \frac{1}{3}(x-2)\cdot 1 \\ &= -\frac{4}{3}x + \frac{20}{3} + \frac{1}{3}x - \frac{2}{3} \\ &= -x + 6. \end{aligned}$$



- To generalize the concept of linear interpolation, consider the construction of a polynomial of degree at most n that passes through the following $n+1$ points:

$$(x_0, f(x_0)), \quad (x_1, f(x_1)), \quad \cdots, \quad (x_n, f(x_n)).$$



- Construct for each $k = 0, 1, \cdots, n$, a function $L_{n,k}(x)$ such that

$$L_{n,k}(x_i) = \delta_{ik} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k. \end{cases}$$

- To satisfy $L_{n,k}(x_i) = 0$ for each $i \neq k$ requires the numerator of $L_{n,k}(x)$ contains

$$(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)$$

- To satisfy $L_{n,k}(x_k) = 1$ requires the denominator of $L_{n,k}(x)$ must be this same term but evaluated at x_k

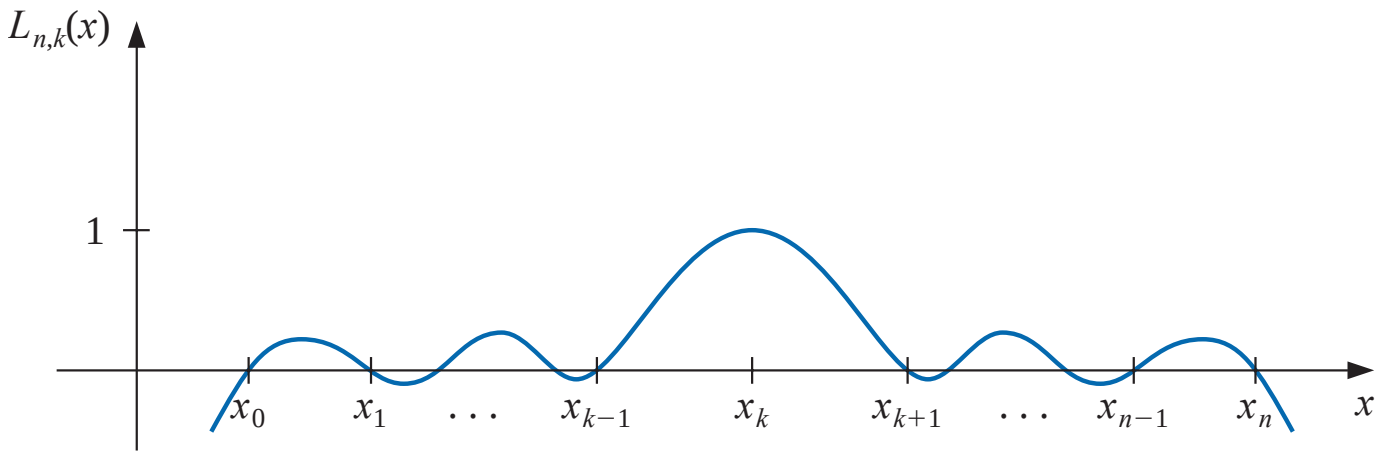
$$(x_k - x_0) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)$$

Definition 3 (Lagrange Interpolating Polynomials).

The n**-th Lagrange interpolating polynomials** are defined by

$$L_{n,k}(x) = \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$$

$$= \prod_{\substack{i\neq k\\ i=0}}^{n} \frac{(x-x_i)}{(x_k-x_i)} \quad \text{for each} \quad k = 0, 1, \cdots, n.$$



We may write $L_{n,k}(x)$ simply as $L_k(x)$ when there is no confusion as to its degree.

Theorem 4.

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

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

This 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 $k = 0, 1, \cdots, n$

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

Example 5.

Use the numbers (called nodes) $x_0 = 2$, $x_1 = 2.75$, and $x_2 = 4$ to find the second-degree Lagrange interpolating polynomial $P_2(x)$ for the function $f(x) = \frac{1}{x}$. Then use this polynomial to approximate $f(3) = \frac{1}{3}$.

Solution (1/2)

- The Lagrange polynomials associated with $x_0 = 2$, $x_1 = 2.75$, and $x_2 = 4$ are

$$
\begin{aligned}
L_0(x) &= \frac{(x-2.75)(x-4)}{(2-2.75)(2-4)} = \frac{2}{3}(x-2.75)(x-4).\\
L_1(x) &= \frac{(x-2)(x-4)}{(2.75-2)(2.75-4)} = -\frac{16}{15}(x-2)(x-4).\\
L_2(x) &= \frac{(x-2)(x-2.75)}{(2.75-2)(4-2.75)} = \frac{2}{5}(x-2)(x-2.75).
\end{aligned}
$$

Solution (2/2)

- Also,

$$f(x_0) = f(2) = \frac{1}{2}, \quad f(x_1) = f(2.75) = \frac{4}{11}, \quad \text{and} \quad f(x_2) = f(4) = \frac{1}{4}$$

- So the second-degree Lagrange Interpolating polynomial is

$$\begin{aligned} P_2(x) &= \sum_{k=0}^{2} f(x_k) L_k(x) \\ &= \frac{1}{2} \cdot \frac{2}{3}(x-2.75)(x-4) - \frac{4}{11} \cdot \frac{16}{15}(x-2)(x-4) \\ &\quad + \frac{1}{4} \cdot \frac{2}{5}(x-2)(x-2.75) \\ &= \frac{1}{22}x^2 - \frac{35}{88}x + \frac{49}{44} \end{aligned}$$

- Use $P_2(x)$ to approximate $f(x)$ at $x = 3$

$$f(3) \approx P_2(3) = \frac{29}{88} = 0.32955. \quad |f(3) - P_2(3)| = 0.03418$$

For the error bound of the Lagrange interpolation, we have the following result.

Theorem 6 (Lagrange Interpolation Error Theorem).

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* $x \in [a, b]$, *there exists a number* $\xi(x)$ *between* $\min\{x_0, x_1 \cdots x_n\}$ *and* $\max\{x_0, x_1 \cdots x_n\}$, *and hence in* $[a, b]$, *such that*

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

where $P(x)$ *is the interpolating polynomial of* $f(x)$.

Comparison of Lagrange polynomials and Taylor polynomials

- The nth-degree Taylor polynomial around x_0 concentrates all the known information at x_0, and has an error term of the form

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

- The nth-degree Lagrange polynomial uses information at distinct numbers $x_0, x_1, \cdots, x_n$, and, in place of $(x-x_0)^n$, its error formula uses a product of the $n+1$ terms:

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

Example 7.

In Example 5, we found the 2nd-degree Lagrange polynomial $P_2(x)$ for $f(x) = \dfrac{1}{x}$ on $[2, 4]$ using the nodes $x_0 = 2, x_1 = 2.75, x_2 = 4$ Determine the error form of this polynomial, and the maximum error when the polynomial is used to approximate $f(x)$ for $x \in [2, 4]$.

Solution (1/2)

- Since $f(x) = \frac{1}{x}$, we have

$$f'(x) = -x^{-2}, \quad f''(x) = 2x^{-3}, \quad f'''(x) = -6x^{-4}.$$

- By Theorem 6, we have on the interval $[2, 4]$,

$$|f(x) - P(x)| = \left|\frac{f'''(\xi)}{3!}g(x)\right| = \frac{6\xi^{-4}}{6}|g(x)| = \frac{1}{\xi^4}|g(x)| \leq \frac{1}{16}|g(x)|$$

where, $g(x) = (x-2)(x-2.75)(x-4) = x^3 - \frac{35}{4}x^2 + \frac{49}{2}x - 22.$

Solution (2/2)

- We now need to determine the maximum value of $|g(x)|$ on $x \in [2, 4]$.

$$g'(x) = 3x^2 - \frac{35}{2}x + \frac{49}{2} = \frac{1}{2}(3x - 7)(2x - 7).$$

- The critical points are $x_1 = \frac{7}{3}$ and $x_2 = \frac{7}{2}$. The global extrema are among the critical points and endpoints $x = 2$ and $x = 4$.

$$g(2) = 0, \quad g(\frac{7}{3}) = \frac{25}{108}, \quad g(\frac{7}{2}) = -\frac{9}{16}, \quad g(4) = 0.$$

- Hence, the maximum error is

$$\max_{x \in [2,4]} |f(x) - P(x)| \leq \frac{1}{16} \left| -\frac{9}{16} \right| = \frac{9}{256} \approx 0.0351526.$$

Note that in Example 5, we found the error at $x = 3$ is

$$|f(3) - P_2(3)| = 0.03418 < 0.0351526 = \max_{2 \leq x \leq 4} |f(x) - P_2(x)|.$$

Coding Exercise (1/2)

Write a MATLAB subroutine

$$L = LagPoly(dataX, k, x)$$

to realize the evaluation of the Lagrange polynomial

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

- the input $dataX = [x_0, x_1, \cdots, x_n]$ contains $n+1$ distinct points.
- the input k is the index of the Lagrange polynomial.
- the input x is the point you want to evaluate.
- the output L is the value of $L_k(x)$.

Coding Exercise (2/2)

Write a second MATLAB subroutine

$$y = LagInterp(dataX, dataY, x)$$

to realize the evaluation of the Lagrange interpolating polynomial

$$P(x) = \sum_{k=1}^{n} y_k L_k(x)$$

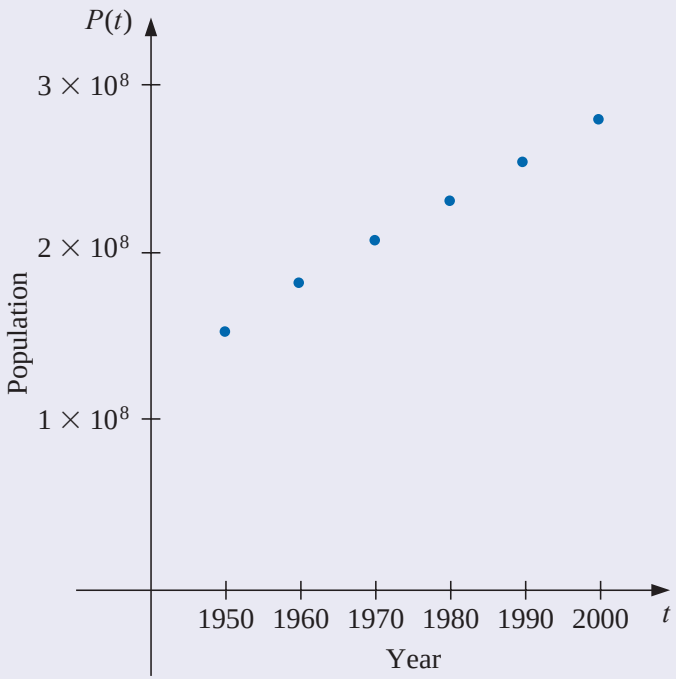
- the input $dataX = [x_0, x_1, \cdots, x_n]$ stores $n+1$ distinct points.
- the input $dataY = [y_0, y_1, \cdots, y_n]$ stores $n+1$ y-values.
- the input x is the point you want to evaluate.
- the output y is the value of $P(x)$.
- for evaluation of $L_k(x)$, you can call the first subroutine

$$L = LagPoly(dataX, k, x)$$

Exercise

A census of the population of the United States is taken every 10 years. The following table lists the population, in thousands of people, from 1950 to 2000, and the data are also represented in the figure.

Year	1950	1960	1970	1980	1990	2000
Population (in thousands)	151,326	179,323	203,302	226,542	249,633	281,422



- In reviewing these data, we might ask whether they could be used to provide a reasonable estimate of the population, say, in 1975 or even in the year 2020.
- We can construct the Lagrange interpolation on these given data to generate a fifth degree polynomial.
- Then use the interpolation polynomial for prediction.

3.2 Divided Differences

- Recall the n-th Lagrange Interpolation $P_n(x)$ of the function $f(x)$ at $n+1$ distinct points $x_0, x_1 \cdots, x_n$.

$$P_n(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x)$$

where

$$L_{n,k}(x) = \prod_{\substack{i \neq k \\ i=0}}^{n} \frac{(x - x_i)}{(x_k - x_i)}.$$

- **Question:** What if we have one more data point available $(x_{n+1}, f(x_{n+1}))$, then how to construct a new $n+1$-th degree interpolation $P_{n+1}(x)$?
- **Answer:** We have to abandon all Lagrange polynomial $L_{n,k}(x)$, and reconstruct new Lagrange polynomials $L_{n+1,k}(x)$.
- **Question:** Is there a more efficient way for adding more data points?

- Although the interpolation polynomial $P_n(x)$ is unique, there are alternative representations that are useful in certain situations.
- In fact, we can write $P_n(x)$ in the following form

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

  for appropriate constants $a_0, a_1, \cdots, a_n$.
- Note that

$$\begin{aligned} P_n(x_0) = f(x_0) \quad &\Longrightarrow \quad a_0 = f(x_0) \\ P_n(x_1) = f(x_1) \quad &\Longrightarrow \quad a_1(x_1 - x_0) = f(x_1) - f(x_0) \\ &\Longrightarrow \quad a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

- **Exercise**: verify that

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

We now introduce the divided-difference notation, which will be very useful in determine the values of a_i.

Definition 8 (Divided Difference).

- The **zeroth divided difference** of the function f at x_i is
$$f[x_i] = f(x_i).$$
- The **first divided difference** of f at x_i and x_{i+1} is
$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$
- The **second divided difference** of f at x_i, x_{i+1} and x_{i+2} is
$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}.$$
- In general, the **k-th divided difference** of f at x_i, x_{i+1}, $\cdots$, x_{i+k} is
$$f[x_i, x_{i+1}, \cdots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \cdots, x_{i+k}] - f[x_i, x_{i+1}, \cdots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

- It can be seen that

$$\begin{aligned} a_0 &= f(x_0) = f[x_0], \\ a_1 &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1] \\ a_2 &= \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0} = f[x_0, x_1, x_2] \end{aligned}$$

- In general, we have

$$a_k = f[x_0, x_1, \cdots, x_k], \quad \text{for all } k = 0, 1, \cdots, n.$$

Interpolation with Newton's Divided Difference

The Lagrange interpolation $P_n(x)$ of $f(x)$ at $x_0, x_1, \cdots, x_n$ can be written as

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

Divided Difference Table

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	$f[x_5]$			

Example 9.

Compute a divided difference table for these function values:

x	3	1	5	6
$f(x)$	1	-3	2	4

Then determine the Newton interpolation polynomial.

Solution (1/2)

Arrange the table vertically to have

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

Solution (2/2)

Note that the values marked in red and blue will be used for constructing the interpolating polynomial

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

The Newton interpolation polynomial is

$$\begin{aligned} P(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 1 + 2(x - 3) + -\frac{3}{8}(x - 3)(x - 1) + \frac{7}{40}(x - 3)(x - 1)(x - 5) \\ &= \frac{1}{40}(7x^3 - 78x^2 + 301x - 350) \end{aligned}$$

Comparison of Lagrange and Newton

- Lagrange Interpolation is sometimes said to require less work, and is sometimes recommended for problems in which it's known, in advance, from previous experience, how many terms are needed for sufficient accuracy.
- The Newton's Interpolation has the advantage that more data points can be added, for improved accuracy, without re-doing the whole problem.

Algorithm for Generating Divided Differences Table

- **Inputs**: **dataX** $= [x_0, x_1, x_2, \cdots, x_n]$, **dataY** $= [y_0, y_1, y_2, \cdots, y_n]$
- **Output**: $\mathbf{C} = [c_{ij}]$ is an $(n+1) \times (n+1)$ divided difference matrix.

$\mathbf{C}$ = divdifTable(**dataX**,**dataY**)

For $j = 1 : n$

  For $i = 0 : n - j$

$$c_{ij} = \frac{c_{i+1,j-1} - c_{i,j-1}}{x_{i+j} - x_i} \quad \text{(note: } c_{ij} = f[x_i, x_{i+1}, \cdots, x_{i+j}]\text{)}$$

  End

End

$$\begin{array}{cc|ccccccc} x_0 & y_0 & c_{00} & c_{01} & c_{02} & c_{03} & \cdots & c_{0,n-1} & c_{0n} \\ x_1 & y_1 & c_{10} & c_{11} & c_{12} & c_{13} & \cdots & c_{1,n-1} & \\ x_2 & y_2 & c_{20} & c_{21} & c_{22} & c_{23} & \cdots & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & & & & \\ x_{n-1} & y_{n-1} & c_{n-1,0} & c_{n-1,1} & & & & & \\ x_n & y_n & c_{n0} & & & & & & \end{array}$$

note: $c_{i,0} = y_i$, for $i = 0, 1, \cdots, n$.

Properties of Divided Difference

- If $z_0, z_1, \cdots, z_n$ is a permutation of $x_0, x_1, \cdots, x_n$, then

$$f[z_0, z_1, \cdots, z_n] = f[x_0, x_1, \cdots, x_n].$$

(**Hint**: consider the leading coefficient of $P_n(x)$.)

- Suppose that $f \in C^n[a, b]$ and $x_0, x_1, \cdots, x_n$ are distinct numbers in $[a, b]$. Then there exists a number $\xi \in (a, b)$ with

$$f[x_0, x_1, \cdots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

(**Hint**: use the generalized Rolle's theorem.)

3.3 Hermite Interpolation

- In previous sections, we introduced the interpolation $P(x)$ of a continuous function $f(x)$ at some given points

x_0	x_1	x_2	$\cdots$	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

- In some applications, we want not only the interpolation function $P(x)$ to match with $f(x)$ at some points, but also the derivatives of $P(x)$ to match with the derivatives of $f(x)$ at those points. These interpolating polynomials are called **osculating polynomials**.

- In particular, we look for a polynomial $P(x)$ of the least degree such that $P(x)$ and $P'(x)$ agrees with f and its derivative f' at $n+1$ distinct points $x_0, x_1, \cdots, x_n$, i.e.,

$$\begin{aligned} P(x_i) &= f(x_i) \\ P'(x_i) &= f'(x_i) \end{aligned} \qquad i = 0, 1, \cdots, n.$$

This is called **Hermite interpolation**.

- Recall the nth-degree **Lagrange interpolating polynomials**:

$$L_{n,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)} \qquad k = 0, 1, \cdots, n.$$

They satisfy $L_{n,k}(x_i) = \delta_{ik}$.

Construction of Hermite Interpolation (1/3)

Define $n + 1$ polynomials:

$$A_k(x) = \Big(1 - 2(x - x_k)L'_{n,k}(x_k)\Big)L^2_{n,k}(x) \qquad k = 0, 1, \cdots, n.$$

It can be verified that

- The degree of A_k is $2n + 1$.
- $A_k(x_i) = \delta_{ik}$.

Construction of Hermite Interpolation (2/3)

- Also, the derivative of $A_k(x)$ is

$$A_k'(x) = \Big(-2L_{n,k}'(x_k)\Big)L_{n,k}^2(x) + \Big(1-2(x-x_k)L_{n,k}'(x_k)\Big)2L_{n,k}(x)L_{n,k}'(x).$$

- Thus,

$$A_k'(x_k) = -2L_k'(x_k) + 2L_k'(x_i) = 0,$$

and

$$A_k'(x_i) = 0, \quad \text{for } i \neq k.$$

- In summary, we have

$$A_k(x_i) = \delta_{ik}, \quad A_k'(x_i) = 0.$$

Construction of Hermite Interpolation (3/3)

Define another $n+1$ polynomials:

$$B_k(x) = (x - x_k)L_{n,k}^2(x) \qquad k = 0, 1, \cdots, n.$$

It can be verified that

- The degree of $B_k(x)$ is $2n+1$.
- $B_k(x_i) = (x_i - x_k)L_{n,k}^2(x_i) = 0 \qquad k = 0, 1, \cdots, n.$

The derivative of $B_k(x)$ is

$$B_k'(x) = L_{n,k}^2(x) + 2(x - x_k)L_{n,k}(x)L_{n,k}'(x).$$

Thus, so that $B_k'(x_k) = 1$, and $B_k'(x_i) = 0$, for $i \neq k$. Hence,

$$B_k(x_i) = 0, \quad B_k'(x_i) = \delta_{ik}.$$

Hermite Interpolation

If $f \in C^1[a,b]$ and $x_0, x_1 \cdots, x_n \in [a,b]$ are distinct, the unique polynomial of least degree agreeing with f and f' at $x_0, x_1 \cdots, x_n$ is the Hermite polynomial of degree at most $2n+1$ given by

$$H_{2n+1}(x) = \sum_{k=0}^{n} f(x_k) A_k(x) + \sum_{k=0}^{n} f'(x_k) B_k(x).$$

where

$$\begin{aligned} A_k(x) &= \Big(1 - 2(x - x_k) L'_{n,k}(x_k)\Big) L^2_{n,k}(x), \\ B_k(x) &= (x - x_k) L^2_{n,k}(x), \end{aligned}$$

and $$L_{n,k}(x) = \prod_{\substack{i \neq k \\ i=0}}^{n} \frac{(x - x_i)}{(x_k - x_i)}.$$

Example 10.

Determine the Hermite polynomial that interpolates the data below

x	1	2
$f(x)$	1	-3
$f'(x)$	-1	2

Solution(1/2)

- First compute the Lagrange polynomials of the point $x_0 = 1$ and $x_1 = 2$.

$$L_0(x) = -(x-2), \quad L_1(x) = x - 1$$

- Next, we compute third-order polynomials ($n = 1,\ 2n+1 = 3$).

$$\begin{aligned}
A_0(x) &= (1 - 2(x-1)(-1))(x-2)^2 = (2x-1)(x-2)^2 \\
A_1(x) &= (1 - 2(x-2))(x-1)^2 = -(2x-5)(x-1)^2 \\
B_0(x) &= (x-1)(x-2)^2, \\
B_1(x) &= (x-2)(x-1)^2
\end{aligned}$$

Solution (2/2)

Then, the Hermite interpolating polynomial is

$$\begin{aligned} H_3(x) &= f(x_0)A_0(x) + f(x_1)A_1(x) + f'(x_0)B_0(x) + f'(x_1)B_1(x) \\ &= 1 \cdot (2x-1)(x-2)^2 + (-3) \cdot -(2x-5)(x-1)^2 \\ &\quad +(-1) \cdot (x-1)(x-2)^2 + 2 \cdot (x-2)(x-1)^2 \\ &= 9x^3 - 39x^2 + 50x - 19. \end{aligned}$$

- There is an alternative approach to construct Hermite interpolation using Newton's divided differences.
- Since each data point x_i is associated with two values $f(x_i)$ and $f'(x_i)$. We denote

$$z_{2i} = z_{2i+1} = x_i, \quad i = 0, 1, \cdots n.$$

and conduct the divided difference table using z_0, z_1, $\cdots$, z_{2n+1}.

- Since $z_{2i} = z_{2i+1} = x_i$, we cannot define $f[z_{2i} = z_{2i+1}]$ by the standard first-divided difference formula

$$f[z_{2i}, z_{2i+1}] = \frac{f[z_{2i+1}] - f[z_{2i}]}{z_{2i+1} - z_{2i}},$$

but instead, the reasonable substitution is

$$f[z_{2i}, z_{2i+1}] = f'(x_i).$$

- The original data table

x_0	x_1	x_2	$\cdots$	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$
$f'(x_0)$	$f'(x_1)$	$f'(x_2)$	$\cdots$	$f'(x_n)$

can be represented by z_i, $0 \le i \le 2n+1$

x_0	x_0	x_1	x_1	x_2	$\cdots$	x_{n-1}	x_n	x_n
z_0	z_1	z_2	z_3	z_4	$\cdots$	z_{2n-1}	z_{2n}	z_{2n+1}
$f(z_0)$	$f(z_1)$	$f(z_2)$	$f(z_3)$	$f(z_4)$	$\cdots$	$f(z_{2n-1})$	$f(z_{2n})$	$f(z_{2n+1})$
$f(x_0)$	$f(x_0)$	$f(x_1)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_{n-1})$	$f(x_n)$	$f(x_n)$

- The only difference is that the first divided difference

$$f[z_0, z_1] = \frac{f[z_1] - f[z_0]}{z_1 - z_0} = \frac{f[x_0] - f[x_0]}{x_0 - x_0} = f'(x_0).$$

- Similar argument applies for $f[z_2, z_3]$, $f[z_4, z_5]$, $\cdots$, $f[z_{2n}, z_{2n+1}]$.

Divided-Difference Table for Hermite Polynomial

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

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

Example 11.

Determine the Hermite polynomial that interpolates the data below

x	1	2
$f(x)$	1	-3
$f'(x)$	-1	2

using Newton's divided differences table.

Solution

The divided difference table is

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

Then the 3rd-degree Hermite interpolating polynomial is

$$\begin{aligned} H_3(x) &= 1 - (x-1) - 3(x-1)^2 + 9(x-1)^2(x-2) \\ &= 9x^3 - 39x^2 + 50x - 19. \end{aligned}$$

For the error bound of the Hermite interpolation, we have the following result.

Theorem 12 (Hermite Interpolation Error Theorem).

Suppose $x_0, x_1, \cdots, x_n$ *are distinct numbers in the interval* $[a, b]$ *and* $f \in C^{2n+2}[a, b]$. *Then, for each* $x \in [a, b]$, *there exists a number* ξ *in* (a, b) *(generally unknown), such that*

$$f(x) = H_{2n+1}(x) + \frac{f^{(2n+2)}(\xi)}{(2n+2)!}(x - x_0)^2(x - x_1)^2 \cdots (x - x_n)^2,$$

where $P_{2n+1}(x)$ *is the Hermite interpolating polynomial of* $f(x)$.

Example 13.

A baseball pitcher throws a fastball from the pitcher's mound to the catcher. Although the distance from the mound to the home plate is 60 feet 6 inches, the ball typically travels about 55 feet 6 inches. Suppose the initial velocity of the ball is 95 miles per hour (mph), and the terminal velocity is 92 mph. Construct a Hermite interpolating polynomial for the data

Time t (in seconds)	0	0.4
Distance d (in feet)	0	55.5
Speed (in mph)	95	92

1. Use the derivative of Hermite polynomial to estimate the speed of the baseball in mph at $t = 0.2$ seconds.
2. Does the maximum velocity of the ball occur at $t = 0$, or does the derivative of the Hermite polynomial have a maximum exceeding 95 mph? If so, does this seem reasonable?

Solution (1/3)

- Convert "miles per hour" to "feet per second" (note 1 mph = 1.46667 fps).
- Rewrite the table as follow

Time t (in seconds)	0	0.4
Distance $d(t)$ (in feet)	0	55.5
Speed $s(t) = d'(t)$(in fps)	139.33	134.93

- Generate the Div-Diff table

$$
\begin{array}{llllll}
t_0 = 0 & f[t_0] = 0 & f[t_0, t_1] = 139.33 & f[t_0, t_1, t_2] = -1.45 & f[t_0, t_1, t_2, t_3] = -20.25 \\
t_1 = 0 & f[t_1] = 0 & f[t_1, t_2] = 138.75 & f[t_1, t_2, t_3] = -9.55 & \\
t_2 = 0.4 & f[t_2] = 55.5 & f[t_2, t_3] = 134.93 & & \\
t_3 = 0.4 & f[t_3] = 55.5 & & &
\end{array}
$$

- The Hermite interpolating polynomial is

$$
\begin{aligned}
d(t) &= 0 + 139.33(t-0) - 1.45(t-0)(t-0) - 20.25(t-0)(t-0)(t-0.4) \\
&= -20.25t^3 + 6.65t^2 + 139.33t.
\end{aligned}
$$

Solution (2/3)

- The speed can be approximated by

$$d(t) = -20.25t^3 + 6.65t^2 + 139.33t$$
$$s(t) = d'(t) = -60.75t^2 + 13.3t + 139.33$$

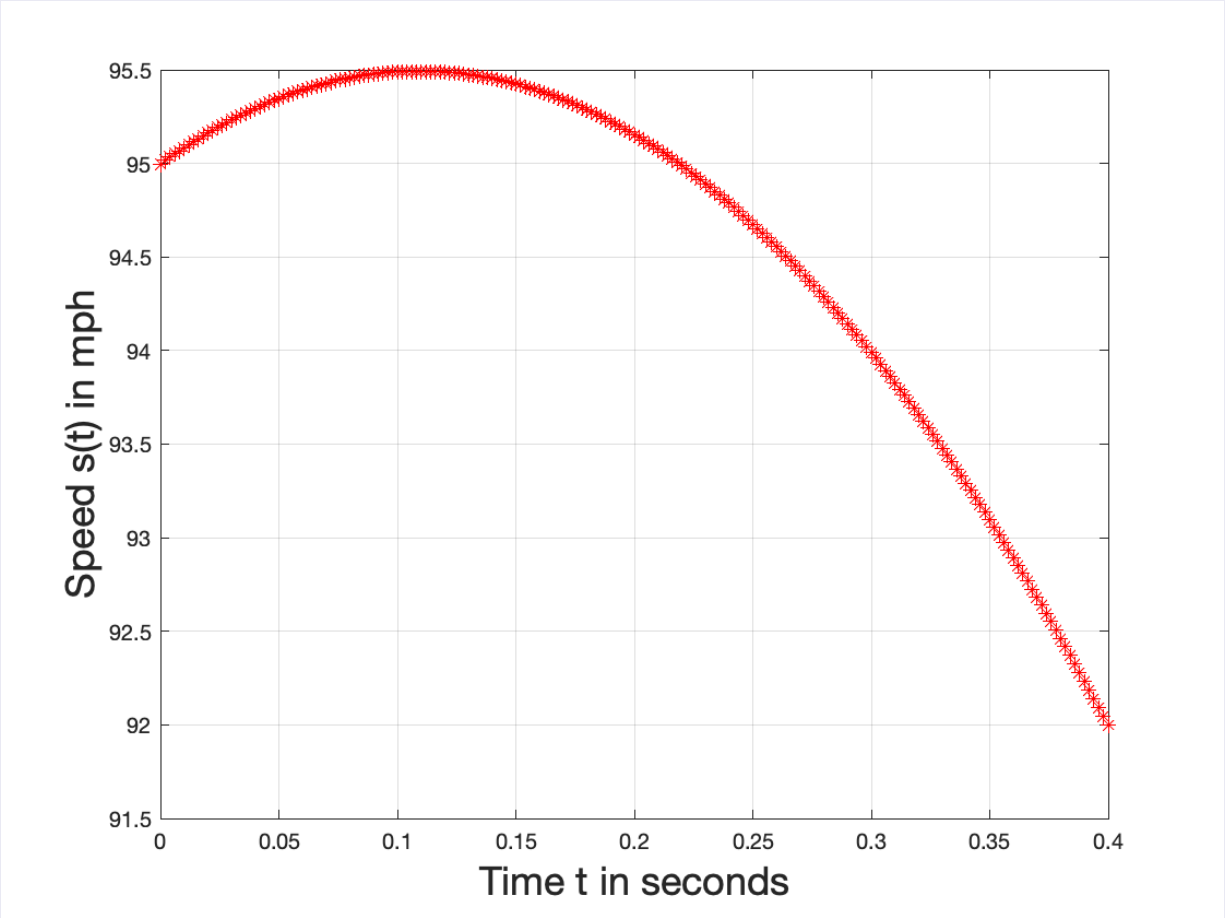
  The speed at $t = 0.2$ is $s(0.2) = 139.56$ fps $= 95.15$ mph.

- To find the maximum speed, we consider the acceleration

$$a(t) = s'(t) = -121.5t + 13.3 = 0 \implies t = 0.1095$$

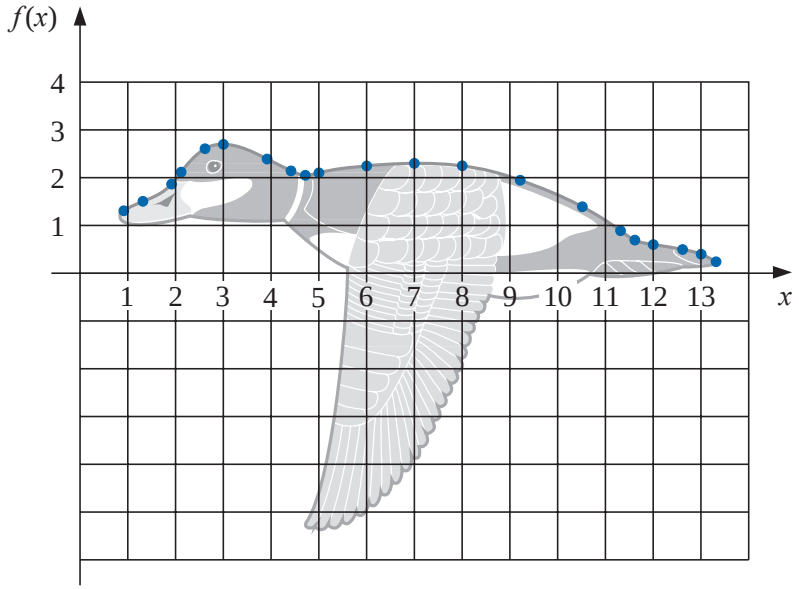
  The maximum speed is $s_{max} = 140.06$ fps = 95.49 mph, occurs at $t = 0.1095$ second.

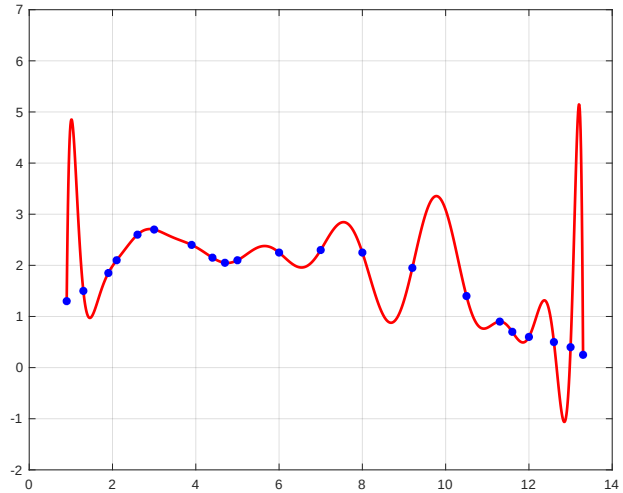
Solution (3/3)


95.5
95
94.5
94
93.5
93
92.5
92
91.5
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
Speed s(t) in mph
Time t in seconds

3.4 Cubic Spline Interpolation

- In previous sections, we introduced the approximation of arbitrary functions on closed intervals using a **single polynomial**.
- However, high-degree polynomials can **oscillate** erratically, that is, a minor fluctuation over a small portion of the interval can induce large fluctuations over the entire range.
- The following is a 20th degree Lagrange interpolation approximating the back of a duck. Clearly this does not reflect the profile of the back.

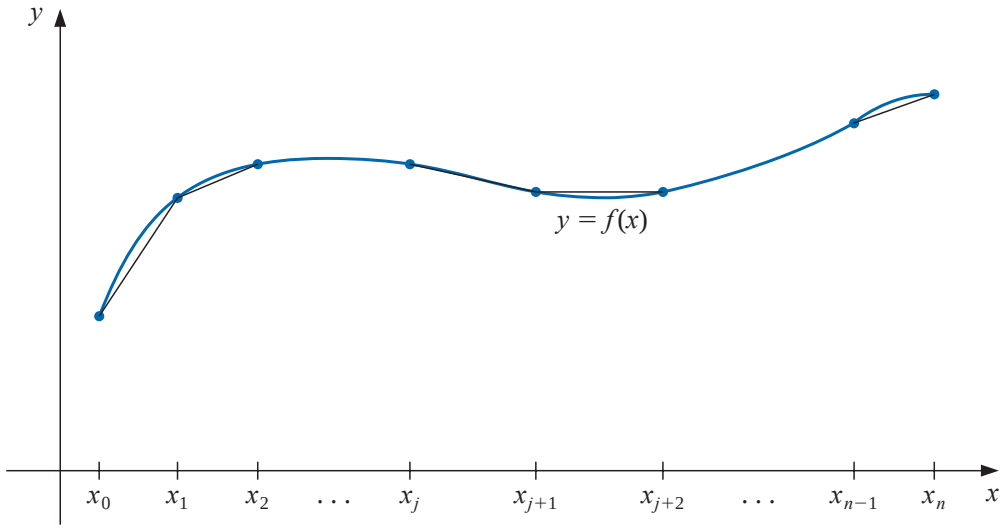




- In this section, we introduce the interpolation using **piecewise polynomials**. This will effectively prevent the oscillation.
- The simplest piecewise-polynomial approximation is **piecewise-linear interpolation**. Consider a set of data points:

x_0	x_1	x_2	$\cdots$	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

- The following is a piecewise linear polynomial interpolation of a smooth curve.

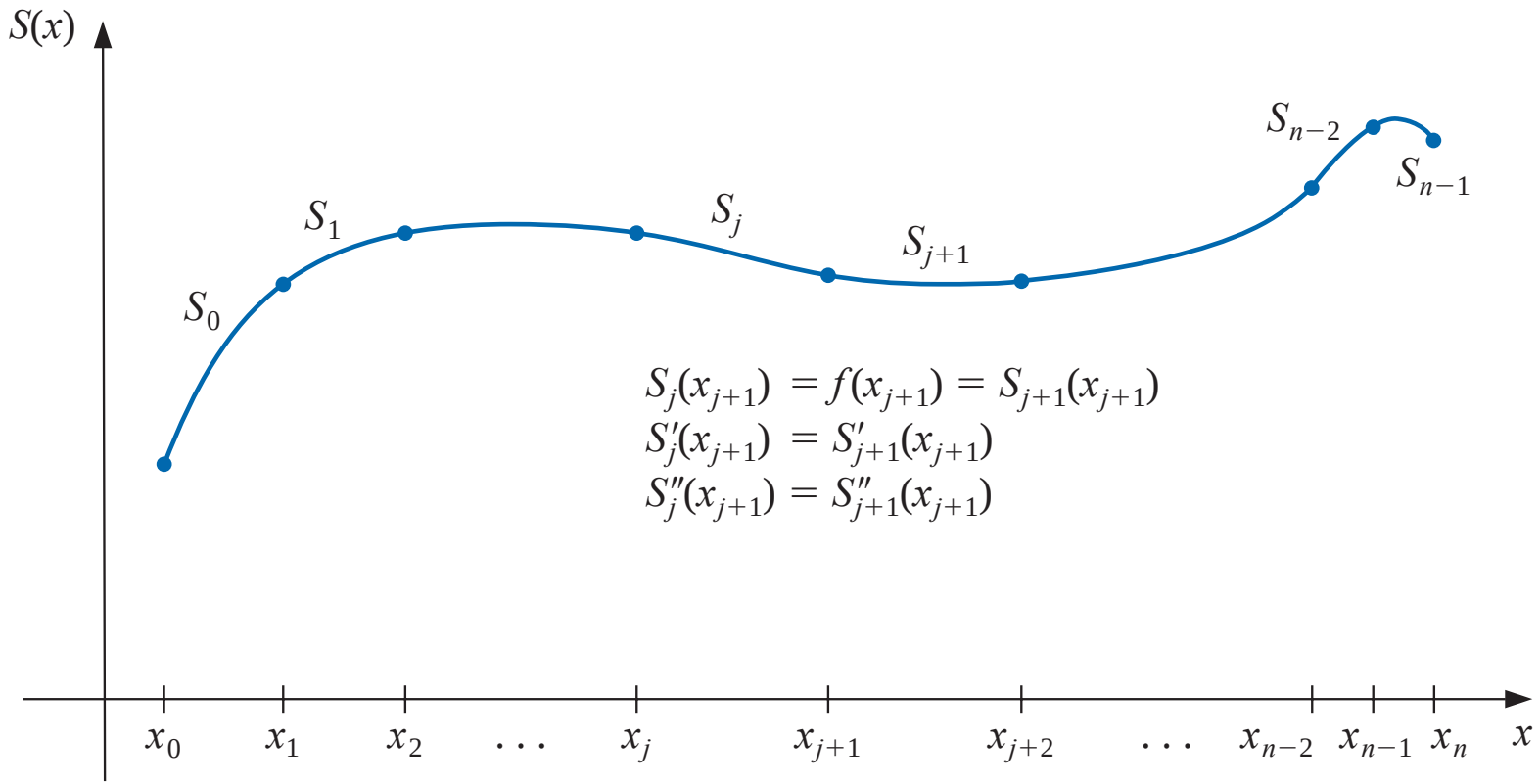


- The **disadvantage** of using piecewise linear approximation is that the approximation function is not ”smooth”, i.e., it is not differentiable at nodes x_i.
- Often it is clear from physical conditions that **smoothness is required**, so the approximating function must be continuously differentiable.
- Using Hermite polynomials could be a choice, however this requires the known of the derivative values at nodes.

The goal in this section is to develop a new interpolation so that

- it uses piecewise polynomials (to avoid oscillation).
- it is continuously differentiable over the entire domain (to ensure the smoothness).
- it requires no specific derivative information of the original function, except perhaps at the two endpoints of the interval (minimum information from original function).

- The most common piecewise-polynomial approximation is called **cubic spline interpolation**.
- The interpolation uses piecewise cubic polynomials, and globally second-order differentiable ($S \in C^2([x_0, x_n])$).



Cubic Spline Interpolation (Natural Spline)

Consider a set of data points:

x_0	x_1	x_2	$\cdots$	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

We construct a cubic spline interpolant $S(x)$ for f satisfies

1. On each subinterval $[x_j, x_{j+1}]$, $S(x)$ is a cubic polynomial, denoted by $S_j(x)$ for $j = 0, 1, \cdots, n-1$.
2. $S_j(x_j) = f(x_j)$ and $S_j(x_{j+1}) = f(x_{j+1})$ for $j = 0, 1, \cdots, n-1$.
3. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ for $j = 0, 1, \cdots, n-2$.
4. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ for $j = 0, 1, \cdots, n-2$.
5. the **natural boundary conditions**, $S''(x_0) = S''(x_n) = 0$.

Example 14.

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

Solution (1/2)

- This spline consists of two cubics. The two subintervals are $[1, 2]$ and $[2, 3]$. We write the piecewise cubic polynomial as follows

$$S(x) = \begin{cases} S_0(x) = a_0 + b_0(x-1) + c_0(x-1)^2 + d_0(x-1)^3, & \text{on } [1,2] \\ S_1(x) = a_1 + b_1(x-2) + c_1(x-2)^2 + d_1(x-2)^3, & \text{on } [2,3]. \end{cases}$$

- There are eight constants to be determined, which requires 8 conditions. Four conditions come from nodal values:

$$\begin{aligned} S_0(1) = 2 &\implies a_0 = 2, \\ S_0(2) = 3 &\implies b_0 + c_0 + d_0 = 1, \\ S_1(2) = 3 &\implies a_1 = 3, \\ S_1(3) = 5 &\implies b_1 + c_1 + d_1 = 2, \end{aligned}$$

Solution (2/2)

- Two conditions come from derivatives at interior nodes $x_1 = 2$.

$$\begin{aligned} S_0'(2) = S_1'(2) &\implies b_0 + 2c_0 + 3d_0 = b_1, \\ S_0''(2) = S_1''(2) &\implies 2c_0 + 6d_0 = 2c_1. \end{aligned}$$

- The last two conditions are from the natural boundary conditions

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

- Solve this system of the eight equations gives the spline

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

Construction of a natural cubic spline

- Assume the following $n+1$ distinct points

x_0	x_1	x_2	$\cdots$	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

  subdivide the interval $[x_0, x_n]$ into n subintervals: $I_j = [x_j, x_{j+1}]$, $j = 0, 1, \cdots, n-1$.

- The cubic spline $S(x)$ restricted on the interval I_j is a cubic polynomial $S_j(x)$ for each $0 \le j \le n-1$:

$$S(x)|_{I_j} = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3.$$

- There are $4n$ constants a_j, b_j, c_j, and d_j to be determined.

- On the interval $[x_j, x_{j+1}]$, denote the length $h_j = x_{j+1} - x_j$. We note that

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

- It is easy to see that

$$a_j = f(x_j), \quad j = 0, 1, \cdots, n-1. \tag{3.1}$$

- If we also let $a_n = f(x_n)$, then by some calculations we can represent b_j and d_j in terms of c_j for $0 \le j \le n-1$:

$$\begin{aligned} b_j &= \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}), \\ d_j &= \frac{1}{3h_j}(c_{j+1} - c_j). \end{aligned} \tag{3.2}$$

Here, we also use the definition $c_n = S_n''(x_n)/2$.

- We obtain a linear system for c_j, $1 \le j \le n-1$:

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

- Write it as a linear system $A\mathbf{x} = \mathbf{b}$ where

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

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

- The matrix A is strictly diagonally dominant, that is,

$$|a_{ii}| > \sum_{j=1}^{n} |a_{ij}|, \quad \text{for all } i = 1, 2, \cdots, n.$$

- The matrix A is nonsingular and the linear system $A\mathbf{x} = \mathbf{b}$ has a unique solution(more in Chapter 6).

- To solve the linear system $A\mathbf{x} = \mathbf{b}$ with Matlab, we can use the command "backslash":

$$\mathbf{x} = A \setminus \mathbf{b}.$$

Matlab File for Natural Spline Coefficient

```
function [a,b,c,d] = natural_spline_coef(dataX,dataY)
%% Cubic Spline (Natual Boundary) Interpolation
%  On each sub-interval, generate four coefficients aj, bj, cj, and dj
%       Sj(x) = aj + bj(x-xj) + cj(x-xj)^2 + dj(x-xj)^3.
%
%  Inputs: dataX dataY are row vectors of same dimension.
%  Output: a is the collum vector that stores {aj} for j=1,2,...,n+1
%          b is the collum vector that stores {bj} for j=1,2,...,n+1
%          c is the collum vector that stores {cj} for j=1,2,...,n+1
%          d is the collum vector that stores {dj} for j=1,2,...,n+1

%% Initialize h
n = length(dataX) - 1;
h = zeros(1,n);
for j = 1:n
    h(j) = dataX(j+1) - dataX(j);
end
%% Prepare for Matrix A
A = zeros(n+1,n+1);
A(1,1) = 1; A(n+1,n+1) = 1;
for j = 2:n
    A(j,j-1) = h(j-1);
    A(j,j+1) = h(j);
    A(j,j) = 2*(h(j-1)+h(j));
end
%% Prepare for vector bb
bb = zeros(n+1,1);
for j = 2:n
    bb(j) = 3/h(j)*(dataY(j+1)-dataY(j)) - 3/h(j-1)*(dataY(j)-dataY(j-1));
end
%% Solve A*c = bb
c = A\bb;
%% Find a,b,d.
a = reshape(dataY(1:n+1),n+1,1);
b = zeros(n,1);
d = zeros(n,1);
for j = 1:n
    b(j) = 1/h(j)*(a(j+1)-a(j)) - h(j)/3*(2*c(j)+c(j+1));
    d(j) = (c(j+1) - c(j))/(3*h(j));
end
%% Remove the last entries of a and c
a(n+1) = [];
c(n+1) = [];
```

Matlab File for Natural Spline Interpolation

```
function y = natural_spline(dataX,dataY,x)
%% Cubic Spline (Natual Boundary) Interpolation
%  On each sub-interval, generate four coefficients aj, bj, cj, and dj
%       Sj(x) = aj + bj(x-xj) + cj(x-xj)^2 + dj(x-xj)^3.
%
%  Inputs: dataX dataY are row vectors of same dimension.
%          x is the query point(s).
%  Output: y is the value of spline interpolation y=S(x) at query points x

%% Generate Spline Coefficient [ai,bi,ci,di] on each interval.
[a,b,c,d] = natural_spline_coef(dataX,dataY);

%% Evaluate the query points
y = zeros(size(x));
for n = 1:length(x)
    for j = 1:length(dataX)-1
        if dataX(j) <= x(n) && dataX(j+1) >= x(n)
            k = j;
            break
        end
    end
    xk = dataX(k);
    y(n) = a(k) + b(k)*(x(n)-xk) + c(k)*(x(n)-xk)^2 + d(k)*(x(n)-xk)^3;
end
```

Example 15.

At the beginning of Chapter 3, we gave some Taylor polynomial to approximate the exponential function $f(x) = e^x$. Use the data points

x	0	1	2	3
$f(x)$	1	e	e^2	e^3

to form a natural cubic spline $S(x)$ that approximates $f(x) = e^x$.

Solution (1/3)

Since this problem involves extensive calculation, we write a Matlab code to solve this problem

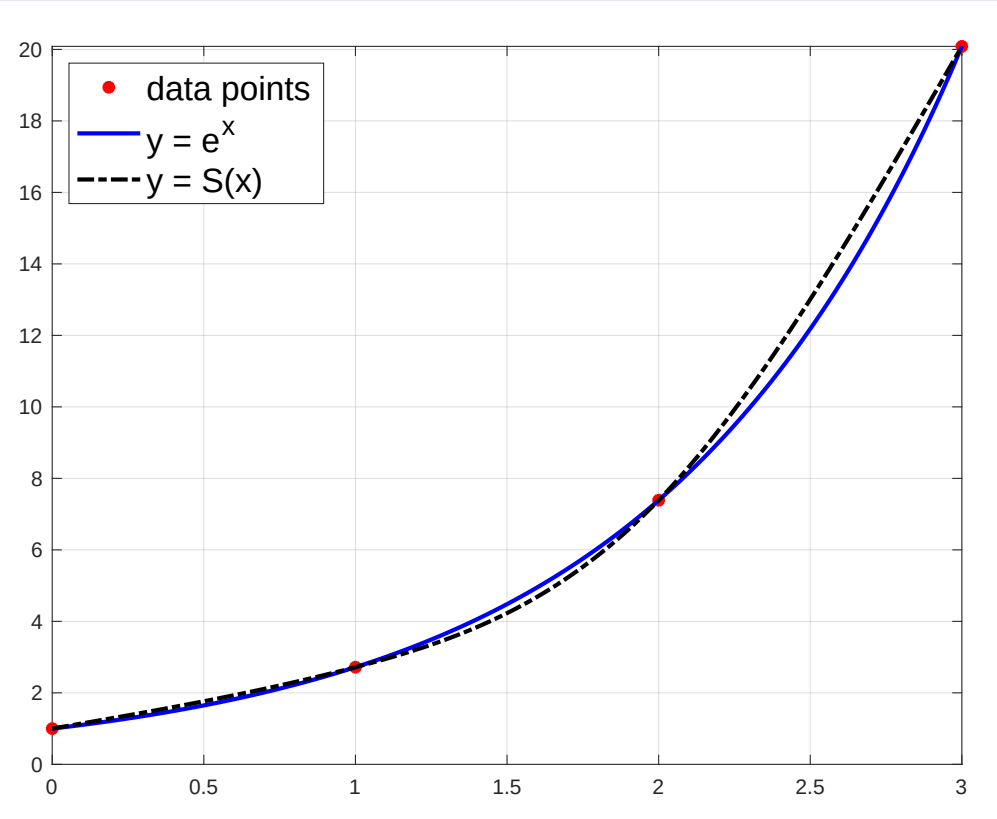
Solution (2/3)

The MATLAB driver file for this example

```
% ex 3.4.1
clc
dataX = [0,1,2,3];
dataY = [exp(0),exp(1),exp(2),exp(3)];

%% generate coefficient of natural spline
[a,b,c,d] = natural_spline_coef(dataX,dataY);
disp('    a         b         c         d')
disp('-----------------------------------------')
disp([a,b,c,d])

%% query point x
x = 0:0.01:3;
y = natural_spline(dataX,dataY,x);

%% plot
figure(1)
plot(dataX,dataY,'r*','linewidth',2)
hold on
plot(x,exp(x),'b-','linewidth',2)
plot(x,y,'k-.','linewidth',2)
lgd = legend('data points','y = e^x','y = S(x)');
hold off
grid on
lgd.FontSize = 16;
lgd.Location = 'NorthWest';
```

Solution (3/3)

$$S(x) = \begin{cases} 1.0000 + 1.4660x + 0.2523x^3, & \text{on } [0,1], \\ 2.7183 + 2.2229(x-1) + 0.7569(x-1)^2 + 1.69107(x-1)^3, & \text{on } [1,2], \\ 7.3891 + 8.8098(x-2) + 5.8301(x-2)^2 - 1.9434(x-2)^3, & \text{on } [2,3]. \end{cases}$$



Clamped Splines

- The **clamped spline** is another type of cubic splines that use different boundary conditions.
- Comparing with the free boundary condition in the natural spline,

$$S''(a) = 0, \quad S''(b) = 0,$$

the clamped spline specifies the slope at the endpoints, i.e.,

$$S'(a) = f'(a), \quad S'(b) = f'(b).$$

- So, the clamped spline requires additional information:

x_0	x_1	x_2	$\cdots$	x_n
$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$
$f'(x_0)$				$f'(x_n)$

- The rest of conditions are exactly the same as the natural splines.

Example 16.

We revisit Example 14, and this time we construct a clamped spline that passes through the points $(1, 2)$, $(2, 3)$, and $(3, 5)$ that has $S'(1) = 2$ and $S'(3) = 1$.

Solution (1/2)

- There are two pieces in the spline $S(x)$:

$$S_0(x) = a_0 + b_0(x-1) + c_0(x-1)^2 + d_0(x-1)^3, \quad \text{on } [1,2]$$
$$S_1(x) = a_1 + b_1(x-2) + c_1(x-2)^2 + d_1(x-2)^3, \quad \text{on } [2,3].$$

- Most conditions (6 out of 8) are the same as the natural spline,

$$f(1) = 2 \implies a_0 = 2, \quad S_0(2) = 3 \implies b_0 + c_0 + d_0 = 1.$$
$$f(2) = 3 \implies a_1 = 3, \quad S_1(3) = 5 \implies b_1 + c_1 + d_1 = 2.$$
$$s_0'(2) = s_1'(2) \implies b_0 + 2c_0 + 3d_0 = b_1.$$
$$s_0''(2) = s_1''(2) \implies 2c_0 + 6d_0 = 2c_1.$$

Solution (2/2)

- The clamped boundary conditions yield

$$s_0'(1) = 2 \implies b_0 = 2,$$

$$s_1'(3) = 1 \implies b_1 + 2c_1 + 3d_1 = 1.$$

- Solve the system for eight unknowns, we have

$$S(x) = \begin{cases} 2 + 2(x-1) - \frac{5}{2}(x-1)^2 + \frac{3}{2}(x-1)^3, & \text{on } [1,2], \\ 3 + \frac{3}{2}(x-1) + 2(x-2)^2 - \frac{3}{2}(x-2)^3, & \text{on } [2,3]. \end{cases}$$

Construction of a clamped cubic spline

- Define the cubic polynomial on each interval to be

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \quad j = 0, 1, \cdots, n-1.$$

  The coefficients $\{a_j\}$, $\{b_j\}$, and $\{d_j\}$ are define as (3.1) and (3.2).

- For $j = 1, 2, \cdots, n-1$ we have the following equations for c_j

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

- In addition, we have the clamped boundary conditions:

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

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

- The clamped spline defined on $a = x_0 < x_1 < \cdots < x_n = b$ is unique. $\{c_n\}$ satisfies the linear system $A\mathbf{x} = \mathbf{b}$ where

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

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

Matlab File for Clamped Spline Coefficient

```
function [a,b,c,d] = clamped_spline_coef(dataX,dataY,dFa,dFb)
%% Cubic Spline (Clamped Boundary) Interpolation
%  On each sub-interval, generate four coefficients aj, bj, cj, and dj
%       Sj(x) = aj + bj(x-xj) + cj(x-xj)^2 + dj(x-xj)^3.
%
%  Inputs: dataX dataY are row vectors of same dimension.
%  Output: a is the collum vector that stores {aj} for j=1,2,...,n+1
%          b is the collum vector that stores {bj} for j=1,2,...,n+1
%          c is the collum vector that stores {cj} for j=1,2,...,n+1
%          d is the collum vector that stores {dj} for j=1,2,...,n+1

%% Initialize h
n = length(dataX) - 1;
h = zeros(1,n);
for j = 1:n
    h(j) = dataX(j+1) - dataX(j);
end

%% Prepare for Matrix A
A = zeros(n+1,n+1);
A(1,1) = 2*h(1); A(1,2) = h(1); % Clamped Boundary Condition
A(n+1,n) = h(n); A(n+1,n+1) = 2*h(n); % Clamped Boundary Condition
for j = 2:n
    A(j,j-1) = h(j-1);
    A(j,j+1) = h(j);
    A(j,j) = 2*(h(j-1)+h(j));
end
%% Prepare for vector bb
bb = zeros(n+1,1);
bb(1) = 3/h(j)*(dataY(2)-dataY(1)) - 3*dFa; % Clamped Boundary Condition
bb(n+1) = 3*dFb - 3/h(n)*(dataY(n+1)-dataY(n)); % Clamped Boundary Condition
for j = 2:n
    bb(j) = 3/h(j)*(dataY(j+1)-dataY(j)) - 3/h(j-1)*(dataY(j)-dataY(j-1));
end
%% Solve A*c = bb
c = A\bb;
%% Find a,b,d.
a = reshape(dataY(1:n+1),n+1,1);
b = zeros(n,1);
d = zeros(n,1);
for j = 1:n
    b(j) = 1/h(j)*(a(j+1)-a(j)) - h(j)/3*(2*c(j)+c(j+1));
    d(j) = (c(j+1) - c(j))/(3*h(j));
end
a(n+1) = [];
c(n+1) = [];
```

Matlab File for Clamped Spline Interpolation

```
function y = clamped_spline(dataX,dataY,dFa,dFb,x)
%% Cubic Spline (Clamped Boundary) Interpolation
%  On each sub-interval, generate four coefficients aj, bj, cj, and dj
%       Sj(x) = aj + bj(x-xj) + cj(x-xj)^2 + dj(x-xj)^3.
%
%  Inputs: dataX dataY are row vectors of same dimension.
%          x is the query point(s).
%  Output: y is the value of spline interpolation y=S(x) at query points x

%% Generate Spline Coefficient [ai,bi,ci,di] on each interval.
[a,b,c,d] = clamped_spline_coef(dataX,dataY,dFa,dFb);

%% Evaluate the query points
y = zeros(size(x));
for n = 1:length(x)
    for j = 1:length(dataX)-1
        if dataX(j) <= x(n) && dataX(j+1) >= x(n)
            k = j;
            break
        end
    end
    xk = dataX(k);
    y(n) = a(k) + b(k)*(x(n)-xk) + c(k)*(x(n)-xk)^2 + d(k)*(x(n)-xk)^3;
end
```

Example 17.

We revisit the spline interpolation of $f(x) = e^x$ at the following points

x	0	1	2	3
$f(x)$	1	e	e^2	e^3.

This time we use the clamped spline with the additional information $f'(0) = 1$, and $f'(3) = e^3$. Then, compare the accuracy with the natural spline interpolation.

Solution (1/3)

Since it involved extensive calculation, we solve the problem using Matlab programing.

Solution (2/4)

```
% ex 3.4.2
clc
dataX = [0,1,2,3];
dataY = [exp(0),exp(1),exp(2),exp(3)];

%% generate coefficient of natural spline
[a,b,c,d] = natural_spline_coef(dataX,dataY);
[a2,b2,c2,d2] = clamped_spline_coef(dataX,dataY,exp(0),exp(3));
disp('Natural Spline')
disp('    a         b         c         d')
disp('-----------------------------------------')
disp([a,b,c,d])
disp('Clamped Spline')
disp('    a         b         c         d')
disp('-----------------------------------------')
disp([a2,b2,c2,d2])

%% query point x
x = 0:0.01:3;
y = natural_spline(dataX,dataY,x);
y2 = clamped_spline(dataX,dataY,exp(0),exp(3),x);

%% plot
figure(1)
plot(dataX,dataY,'r*','linewidth',2)
hold on
plot(x,exp(x),'b-','linewidth',2)
plot(x,y,'k-.','linewidth',2)
plot(x,y2,'g-.','linewidth',2)
lgd = legend('data points','y = e^x','Natural','Clamped');
hold off
grid on
lgd.FontSize = 16;
lgd.Location = 'NorthWest';
```

Solution (3/4)

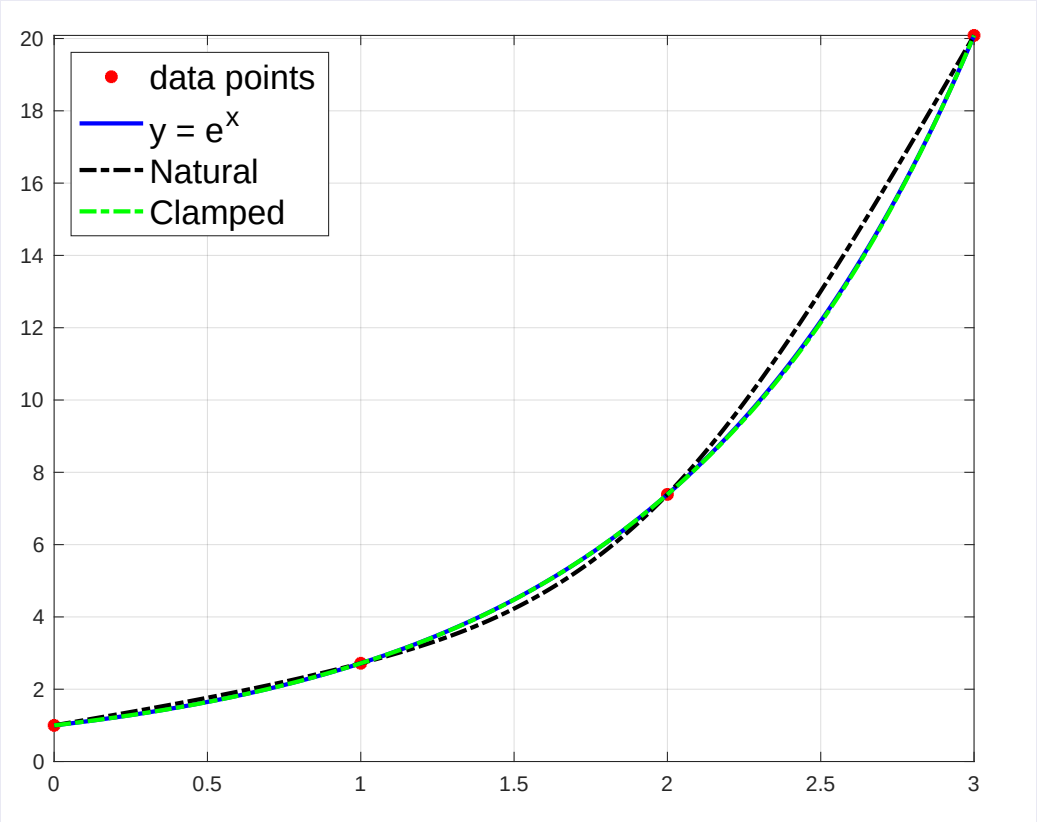
- The natural spline is

$$S(x) = \begin{cases} 1.0000 + 1.4660x + 0.2523x^3, & \text{on } [0,1], \\ 2.7183 + 2.2229(x-1) + 0.7569(x-1)^2 + 1.6911(x-1)^3, & \text{on } [1,2], \\ 7.3891 + 8.8098(x-2) + 5.8301(x-2)^2 - 1.9434(x-2)^3, & \text{on } [2,3]. \end{cases}$$

- The clamped spline is

$$S(x) = \begin{cases} 1.0000 + 1.0000x + 0.4447x^2 + 0.2736x^3, & \text{on } [0,1], \\ 2.7183 + 2.7102(x-1) + 1.2655(x-1)^2 + 0.6951(x-1)^3, & \text{on } [1,2], \\ 7.3891 + 7.3265(x-2) + 3.3509(x-2)^2 + 2.0191(x-2)^3, & \text{on } [2,3]. \end{cases}$$

Solution (4/4)



- From the plot, we can see that the clamped spline is more accurate than the natural spline.
- This is not surprise since the boundary conditions for the clamped spline are exact.

Theorem 18 (Error bound for Clamped Cubic Splines).

Let $f \in C^4[a,b]$ *with*

$$\max_{a \le x \le b} |f^{(4)}(x)| = M.$$

If $S(x)$ *is the unique clamped cubic spline interpolation to* f *with respect to the nodes* $a = x_0 < x_1 < \cdots < x_n = b$*, then for all* $x \in [a,b]$*,*

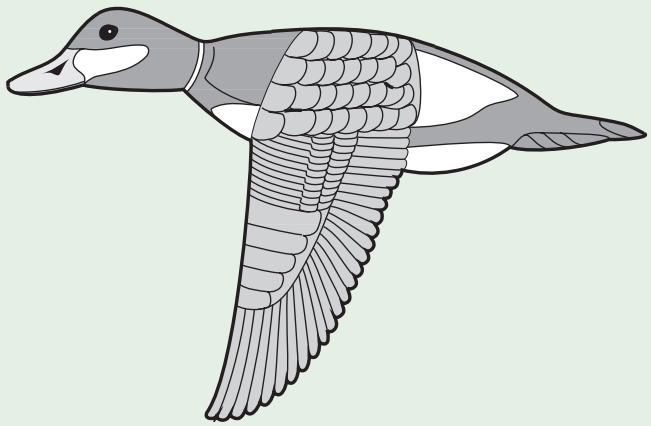
$$|f(x) - S(x)| \le \frac{5M}{384} \max_{0 \le j \le n-1} (x_{j+1} - x_j)^4.$$

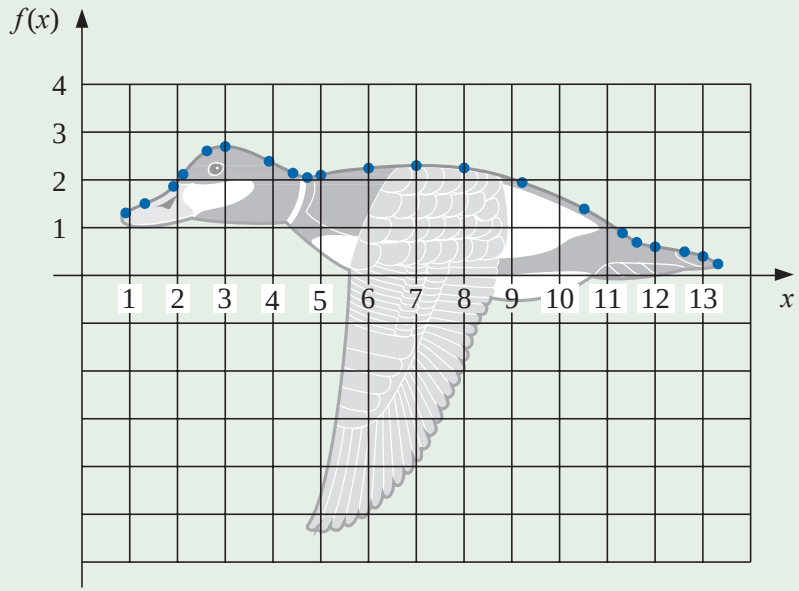
Remark

- A fourth-order error bound also holds in the case of natural spline interpolation, but it is more difficult to express.
- There are other cubic spline interpolations that do not require the derivative of f. For example, the popular "not-a-knot spline" requires that the third-order derivative $S'''(x)$ is continuous at x_1 and x_{n-1}.

Example 19.

In this example, we approximate the top profile of the duck using cubic spline interpolation.





In general, the more points we use, the better approximation we can expect. We choose 21 data points as depicted above and shown below in the table. Note that more points are placed where the curve is changing more rapidly.

x	0.9	1.3	1.9	2.1	2.6	3.0	3.9	4.4	4.7	5.0	6.0	7.0	8.0	9.2	10.5	11.3	11.6	12.0	12.6	13.0	13.3
$f(x)$	1.3	1.5	1.85	2.1	2.6	2.7	2.4	2.15	2.05	2.1	2.25	2.3	2.25	1.95	1.4	0.9	0.7	0.6	0.5	0.4	0.25

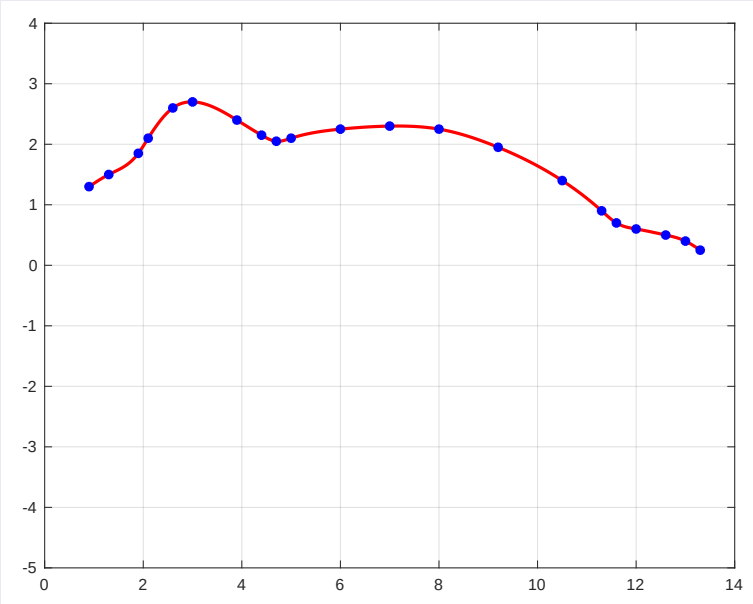
Solution (1/4)

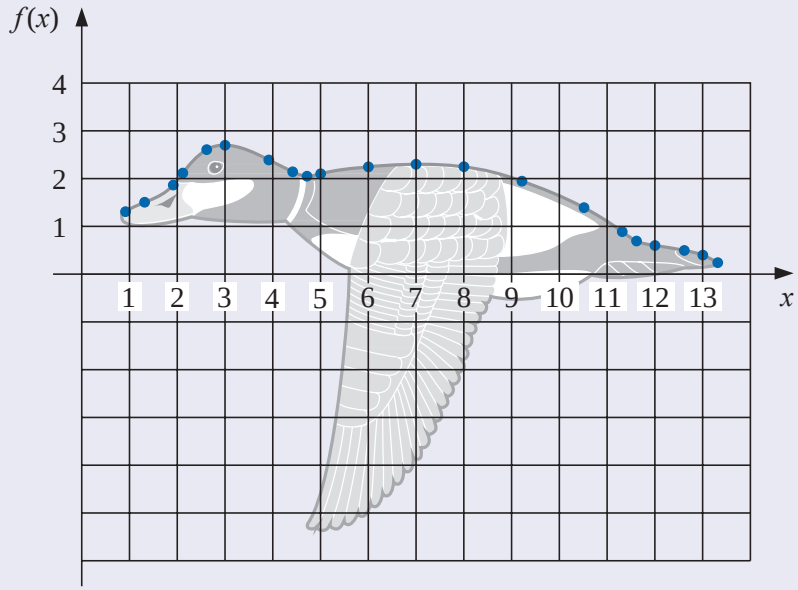
- Since we don't have derivative information, we use the natural spline interpolation.
- We write a Matlab driver file for this example.

```
% ex 3.5.illustration
clear; clc;

%% Input
dataX = [0.9 1.3,1.9 2.1 2.6 3.0 3.9 4.4 4.7 5.0 6.0 7.0 8.0 ...
    9.2 10.5 11.3 11.6 12.0 12.6 13.0 13.3];
dataY = [1.3 1.5 1.85 2.1 2.6 2.7 2.4 2.15 2.05 2.1 2.25 2.3 2.25 ...
    1.95 1.4 0.9 0.7 0.6 0.5 0.4 0.25];
[a,b,c,d] = natural_spline_coef(dataX,dataY); % coefficent of cubic spline

%% Display Outputs
disp('-------------------------------------------------------')
disp(' j      a           b           c           d')
disp('-------------------------------------------------------')
formatSpec = '%2i    % .5f    % .5f    % .5f    % .5f   \n';
fprintf(formatSpec,[(0:length(dataX)-2)',a,b,c,d]')

%% Plot Spline
x = 0.9:0.01:13.3; % query points
y = natural_spline(dataX,dataY,x);
figure(1)
clf;
plot(x,y,'r-','linewidth',2)
hold on
plot(dataX,dataY,'b*','linewidth',2)
axis([0,14,-6,4])
grid on
hold off
```

Solution (2/4)

- Plotting the natural spline interpolation, we observe that the spline curve is accurately recovers the top profile of the duck.





- To use a clamped spline we would need derivative approximations for the endpoints. Even if these approximations were available, we could expect little improvement because of the close agreement of the natural cubic spline to the curve of the top profile.

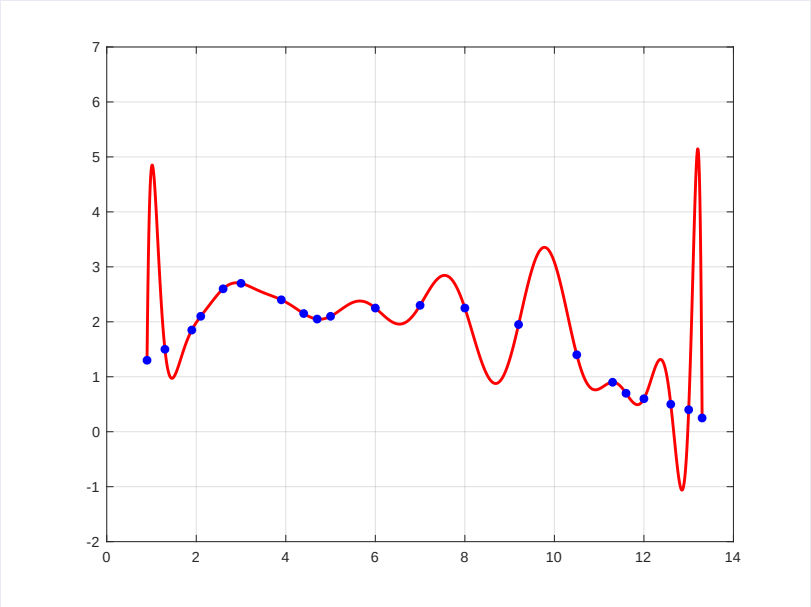
Solution (3/4)

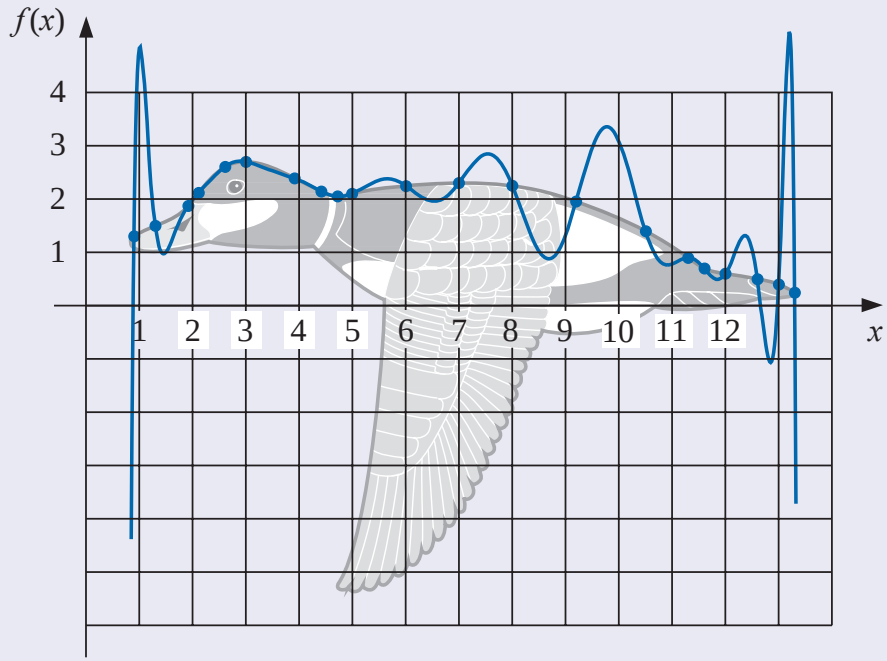
For comparison, we also use Lagrange Interpolation.

```
% Duck Interpolation Comparison
clear; clc;

%% Input
dataX = [0.9 1.3,1.9 2.1 2.6 3.0 3.9 4.4 4.7 5.0 6.0 7.0 8.0 ...
    9.2 10.5 11.3 11.6 12.0 12.6 13.0 13.3];
dataY = [1.3 1.5 1.85 2.1 2.6 2.7 2.4 2.15 2.05 2.1 2.25 2.3 2.25 ...
    1.95 1.4 0.9 0.7 0.6 0.5 0.4 0.25];

%% Find Spline and Lagrange Interpolation
x = 0.9:0.01:13.3; % query points
y = natural_spline(dataX,dataY,x);
yy = LagrangeInterpolation(dataX,dataY,x);

%% Plot Interpolations
figure(1)
clf;
plot(dataX,dataY,'b*','linewidth',2)
hold on
plot(x,y,'r-','linewidth',2)
plot(x,yy,'k-','linewidth',2)
axis([0,14,-2,7])
hold off
grid on
lgd = legend('Data Point','Natural Spline','Lagrange');
lgd.FontSize = 16;
lgd.Location = 'NorthWest';
```

Solution (4/4)

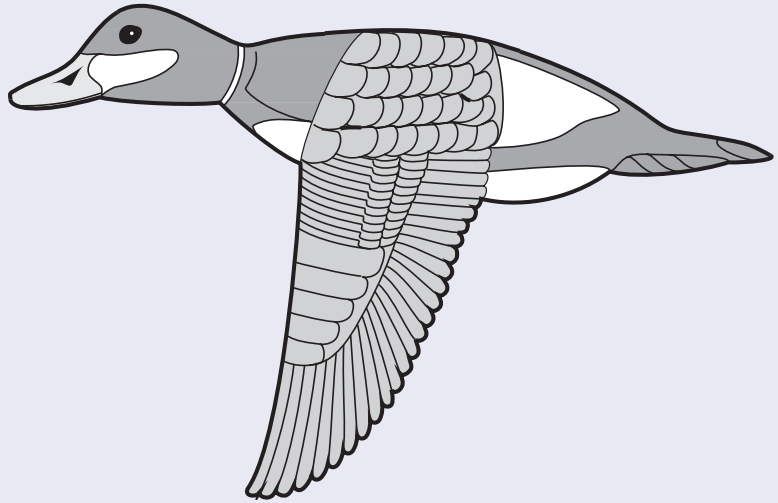
- Plotting the Lagrange interpolation, we observe that the 20th-degree polynomial oscillates wildly. It produces a very strange illustration of the back of a duck.





- This example shows the superiority of the cubic spline interpolation.

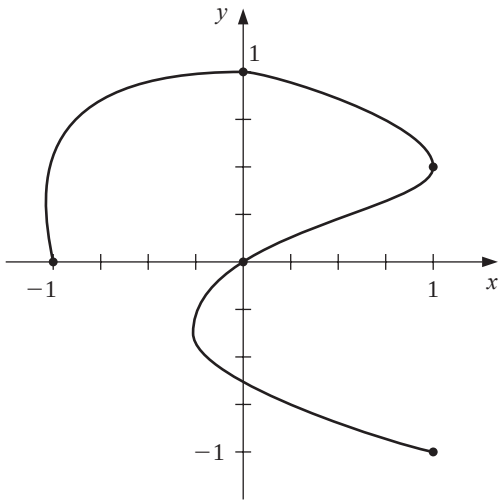
Remark



- Constructing a cubic spline to approximate the lower profile of the duck would be more difficult since the curve for this portion cannot be expressed as a function of x, and at certain points the curve does not appear to be smooth.
- These problems can be resolved by using separate splines to represent various portions of the curve, but a more effective approach to approximating curves of this type is considered in the next section.

3.5 Parametric Curves

- The techniques we developed so far in this chapter cannot be used to generate curves of the form shown below because this curve cannot be expressed as a function $y = f(x)$.



- In this section we will see how to represent general curves (even some hand-drawn curves) using parametric forms.
- This technique can be extended to represent general curves/surfaces in computer graphics.

- Given a set of data points

x_0	x_1	x_2	$\cdots$	x_n
y_0	y_1	y_2	$\cdots$	y_n

  we can use a parameter t, and construct polynomial or piecewise polynomial approximation for

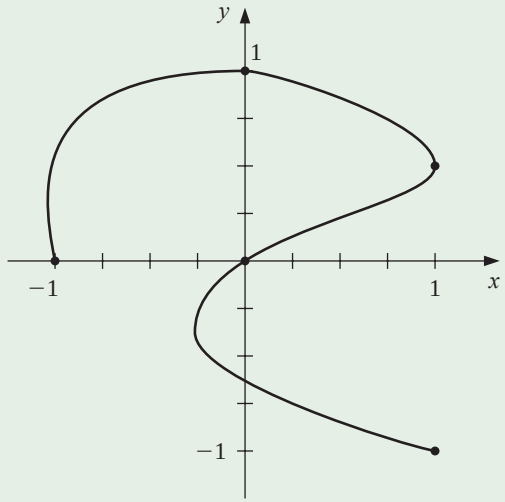
$$x = x(t), \quad \text{and} \quad y = y(t).$$

- To do this, we specify an interval $[t_0, t_n]$, with $t_0 < t_1 < \cdots < t_n$, and construct two approximation functions with

$$x_i = x(t_i), \quad \text{and} \quad y_i = y(t_i), \quad i = 0, 1, \cdots, n.$$

Example 20.

Construct a pair of Lagrange polynomials to approximate the curve show below



Solution (1/3)

There are five points, so we choose the points $\{t_i\}_{i=0}^{4}$ equally spaced in [0,1]:

i	0	1	2	3	4
t_i	0	0.25	0.5	0.75	1
x_i	1	0	1	0	−1
y_i	−1	0	0.5	1	0

Solution (2/3)

We write a MATLAB Driver file for this example

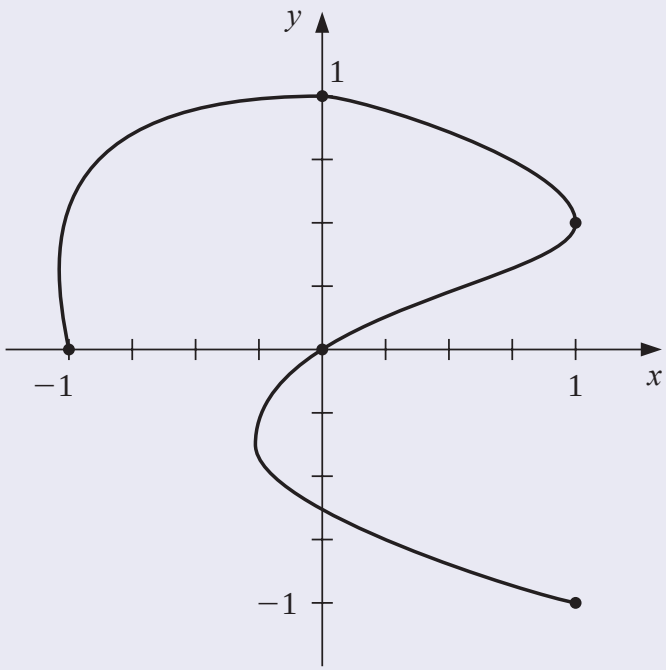
```
% ex3_6_1
dataT = [0,0.25,0.5,0.75,1];
dataX = [-1,0,1,0,1];
dataY = [0,1,0.5,0,-1];

qt = 0:0.001:1;
qx = LagrangeInterpolation(dataT,dataX,qt);
qy = LagrangeInterpolation(dataT,dataY,qt);

figure(1); clf
plot(dataX,dataY,'r*','linewidth',2)
hold
plot(qx,qy,'b-.','linewidth',2)
grid on
lgd = legend('Data Point','Lagrange');
lgd.FontSize = 16;
lgd.Location = 'NorthEast';
```

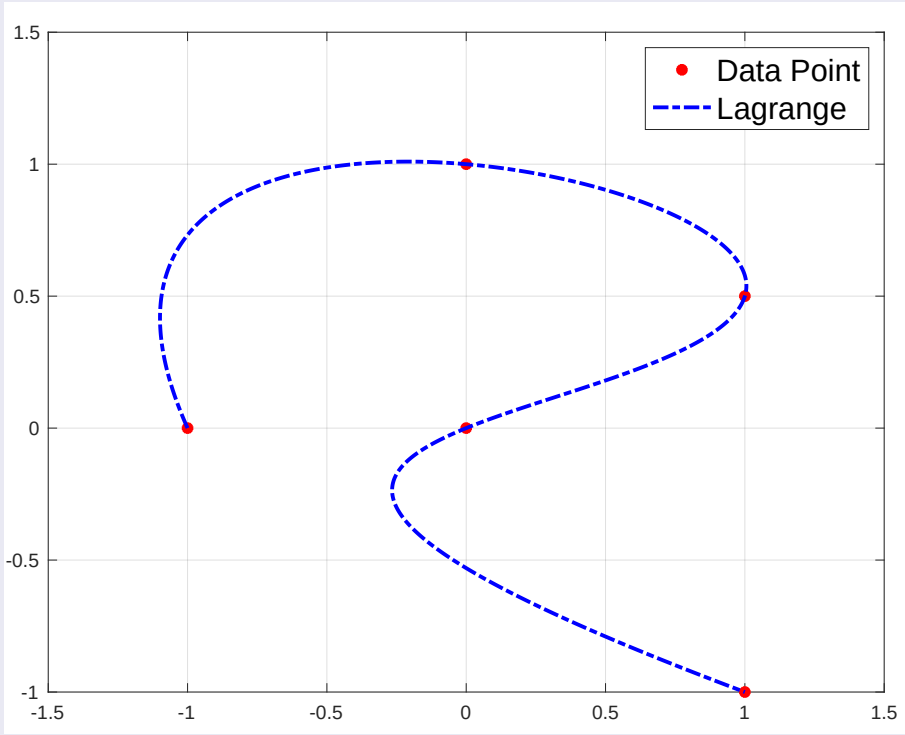
The Interpolation polynomials are

$$x(t) = 64t^4 - \frac{352}{3}t^3 + 60t^2 - \frac{14}{3}t - 1,$$
$$y(t) = -\frac{64}{3}t^4 + 48t^3 - \frac{116}{3}t^2 + 11t.$$

Solution (3/3)

Plotting the parametric system produced the graph below:

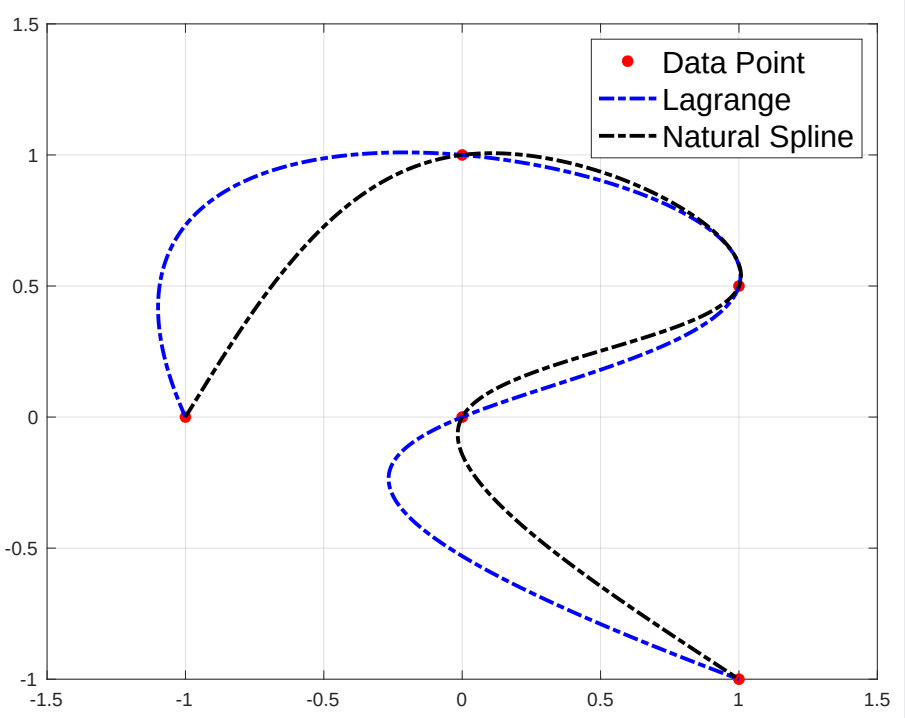




Remark

For this example, we can also use the natural cubic Spline interpolation for the parametric system.

```
% ex3_6_1_2
dataT = [0,0.25,0.5,0.75,1];
dataX = [-1,0,1,0,1];
dataY = [0,1,0.5,0,-1];

qt = 0:0.001:1;

qx = LagrangeInterpolation(dataT,dataX,qt);
qy = LagrangeInterpolation(dataT,dataY,qt);

qxS = natural_spline(dataT,dataX,qt);
qyS = natural_spline(dataT,dataY,qt);

figure(1); clf
plot(dataX,dataY,'r*','linewidth',2)
hold
plot(qx,qy,'b-.','linewidth',2)
plot(qxS,qyS,'k-.','linewidth',2)
grid on
lgd = legend('Data Point','Lagrange','Natural Spline');
lgd.FontSize = 16;
lgd.Location = 'NorthEast';
```



Example 21.

In this example, we demonstrate using natural cubic spline to interpolate arbitrary hand-drawn curve. We need to use the graphic input MATLAB command

[X,Y]=ginput(N)

to get N points from the click of the mouse.
(for more details, type help ginput in MATLAB command window)

A MATLAB Driver File for Parametric Curve

```
% ex3_6_2
clc
clear
close all

%% Input Points on a region [0,1] X [0,1]
figure(1)
grid on
axis([0,1,0,1])
N = 20;
dataX = zeros(N,1); dataY = zeros(N,1);
dataT = (0:1/(N-1):1)';

for n = 1:N
    [X,Y] = ginput(1);
    plot(X,Y,'r*');
    text(X,Y,int2str(n));
    grid on
    axis([0,1,0,1])
    hold on
    dataX(n) = X; dataY(n) = Y;
end
disp(' ')
disp('       I      dataT     dataX     dataY')
disp('-------------------------------------')
disp([(1:N)',dataT,dataX,dataY])

%% Query Points
qt = 0:0.001:1; % use 1000 query points
qx = natural_spline(dataT,dataX,qt);
qy = natural_spline(dataT,dataY,qt);

figure(2);clf
plot(dataX,dataY,'r*',qx,qy,'k-','linewidth',2)
grid on
axis([0,1,0,1])
```

Remarks

- Applications in computer graphics require the rapid generation of smooth curves that can be easily and quickly modified. For both aesthetic and computational reasons, changing one portion of these curves should have little or no effect on other portions of the curves.
- This eliminates the use of interpolating polynomials and splines since changing one portion of these curves affects the whole curve.
- The choice of curve for use in computer graphics is generally a form of the piecewise cubic Hermite polynomial. Popular graphics programs using Hermite cubics are described as **Bézier polynomials**, which uses the "guidepoint" to compute the derivatives at the endpoints in each interval.