Numerical Analysis

10th ed

R L Burden, J D Faires, and A M Burden

Beamer Presentation Slides
Prepared by
Dr. Annette M. Burden
Youngstown State University

July 3, 2015

Weierstrass Theorem Illustration

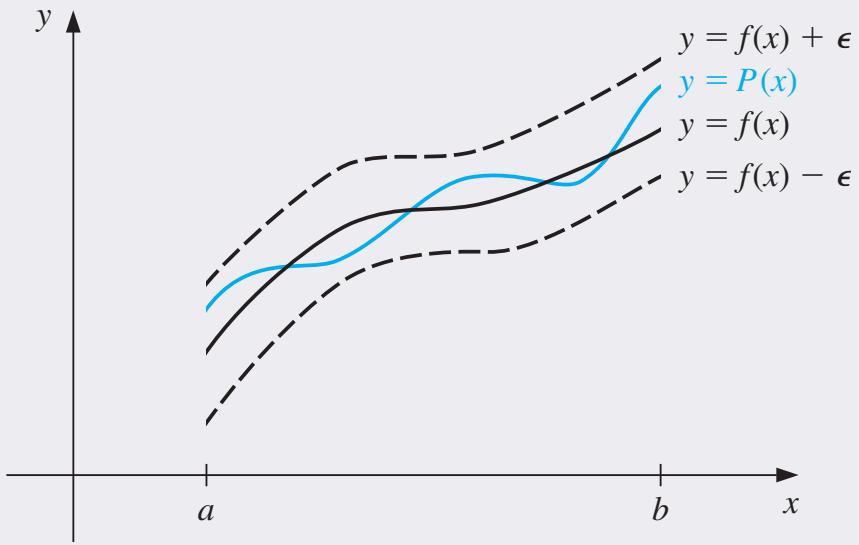

Figure: Figure 3.1

Chapter 3.1: Interpolation and Lagrange Poly

Theorem (3.1 Weierstrass Approximation Theorem)

Suppose that f is defined and continuous on $[a, b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$, with the property that

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

Theorem (3.2: n^{th} Lagrange Interpolating Polynomial)

If $x_0, x_1, \ldots, x_n$ 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

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

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 $k = 0, 1, \ldots, n$,

$$L_{n,k}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{(x-x_i)}{(x_k-x_i)}.$$

The YouTube video developed by Oscar Veliz can serve as a good illustration of the Lagrange Interpolating Polynomial for students. ▸ Lagrange Video

Theorem (3.3)

Suppose $x_0, x_1, \ldots, x_n$ *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* $\min\{x_0, x_1, \ldots, x_n\}$, *and the* $\max\{x_0, x_1, \ldots, x_n\}$ *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 in Theorem 3.2.*

Definition (3.4)

Let f be a function defined at $x_0, x_1, x_2, \ldots, x_n$, and suppose that $m_1, m_2, \ldots, m_k$ are k distinct integers, with $0 \leq m_i \leq n$ for each i. The Lagrange polynomial that agrees with $f(x)$ at the k points $x_{m_1}, x_{m_2}, \ldots, x_{m_k}$ is denoted $P_{m_1,m_2,\ldots,m_k}(x)$.

Theorem (3.5)

Let f be defined at $x_0, x_1, \ldots, x_k$, 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 $x_0, x_1, \ldots, x_k$.

Algorithm 3.1: NEVILLE'S ITERATED INTERPOLATION

To evaluate the interpolating polynomial P on the $n+1$ distinct numbers $x_0, \ldots, x_n$ at the number x for the function f:

INPUT numbers $x, x_0, x_1, \ldots, x_n$; values $f(x_0), f(x_1), \ldots, f(x_n)$ as 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.

The YouTube video developed by Alissa Granger can serve as a good illustration of the Neville's Iterated Interpolation for students. Neville's Video

Algorithm 3.2: NEWTON'S DIVIDED DIFFERENCE

To obtain the divided-difference coefficients of the interpolatory polynomial P on the $(n+1)$ distinct numbers $x_0, x_1, \ldots, x_n$ for the function f:

INPUT $x_0, x_1, \ldots, x_n$; 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 $F_{i,j} = \dfrac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$. $\quad (F_{i,j} = f[x_{i-j}, \ldots, x_i].)$

Step 2 OUTPUT $(F_{0,0}, F_{1,1}, \ldots, F_{n,n})$;
STOP.

The YouTube video developed by Sujoy Krishna Das can serve as a good illustration of the Newton's Divided Difference Formula. ▸ Newton's Video

Theorem (3.6)

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

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

Newton's Forward-Difference Formula

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0).$$

Definition (3.7)

Given the sequence $\{p_n\}_{n=0}^{\infty}$, define the backward difference ∇p_n (read *nabla* p_n) by

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

Higher powers are defined recursively by

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

Newton's Backward-Difference Formula

$$P_n(x) = f[x_n] + \sum_{k=1}^{n} (-1)^k \binom{-s}{k} \nabla^k f(x_n)$$

The YouTube video developed by Umasankar Dhulipalla can serve as a good illustration of the Backward Divided Difference Formula. ▸ Newton's Bsckward-Divided Difference Video

The YouTube video developed by Umasankar Dhulipalla can serve as a good illustration of the Forward Divided Difference Formula. ▸ Newton's Fowrard-Divided Difference Video

Newton's forward and backward difference formulas are not appropriate for approximating $f(x)$ when x lies near the center of the table since neither permits the highest-order difference to have x_0 close to x. A number of divided-difference formulas are available for this case. These methods are known as **centered-difference formulas**. We consider only one Centered-difference formula, Stirling's method.

Definition (Centered Difference: Stirling's Formula)

$$
\begin{aligned}
P_n(x) \quad = \quad & P_{2m+1}(x) = f[x_0] + \frac{sh}{2}(f[x_{-1}, x_0] + f[x_0, x_1]) \\
& + s^2h^2 f[x_{-1}, x_0, x_1] + \frac{s(s^2-1)h^3}{2} f[x_{-2}, x_{-1}, x_0, x_1] \\
& + f[x_{-1}, x_0, x_1, x_2]) + \cdots \\
& + s^2(s^2-1)(s^2-4)\cdots(s^2-(m-1)^2)h^{2m} f[x_{-m}, \ldots, x_m] \\
& + \frac{s(s^2-1)\cdots(s^2-m^2)h^{2m+1}}{2}(f[x_{-m-1}, \ldots, x_m] \\
& + f[x_{-m} \ldots, x_{m+1}]),
\end{aligned}
$$

if $n = 2m + 1$ is odd. If $n = 2m$ is even, we use the same formula but delete the last line.

If $n = 2m + 1$ is odd. If $n = 2m$ is even, we use the same formula but delete the last line. The entries used for this formula are underlined in Table 3.13. The entries used for Stirling's formula are underlined in this table.

Table 3.12

x	$f(x)$	First divided differences	Second divided differences	Third divided differences	Fourth divided differences
x_{-2}	$f[x_{-2}]$				
		$f[x_{-2}, x_{-1}]$			
x_{-1}	$f[x_{-1}]$		$f[x_{-2}, x_{-1}, x_0]$		
		$\underline{f[x_{-1}, x_0]}$		$\underline{f[x_{-2}, x_{-1}, x_0, x_1]}$	
x_0	$\underline{f[x_0]}$		$\underline{f[x_{-1}, x_0, x_1]}$		$\underline{f[x_{-2}, x_{-1}, x_0, x_1, x_2]}$
		$\underline{f[x_0, x_1]}$		$\underline{f[x_{-1}, x_0, x_1, x_2]}$	
x_1	$f[x_1]$		$f[x_0, x_1, x_2]$		
		$f[x_1, x_2]$			
x_2	$f[x_2]$				

Definition (3.8)

Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct numbers in $[a, b]$ and for $i = 0, 1, \ldots, n$ let m_i be a nonnegative integer. Suppose that $f \in C^m[a, b]$, where $m = \max_{0 \le i \le n} m_i$.
The **osculating polynomial** approximating f is the polynomial $P(x)$ of least degree such that

$$\frac{d^k P(x_i)}{dx^k} = \frac{d^k f(x_i)}{dx^k}, \text{ for each } i = 0, 1, \ldots, n \text{ and } k = 0, 1, \ldots, m_i.$$

Theorem (3.9)

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

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x),$$

where, for $L_{n,j}(x)$ denoting the jth Lagrange coefficient polynomial of degree n, we have

$$H_{n,j}(x) = [1 - 2(x - x_j)L'_{n,j}(x_j)]L^2_{n,j}(x) \quad \text{and} \quad \hat{H}_{n,j}(x) = (x - x_j)L^2_{n,j}(x).$$

Moreover, if $f \in C^{2n+2}[a,b]$, then

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

for some (generally unknown) $\xi(x)$ in the interval (a,b).

Algorithm 3.3: HERMITE INTERPOLATION

To obtain the coefficients of the Hermite interpolating polynomial $H(x)$ on the $(n+1)$ distinct numbers $x_0, \ldots, x_n$ for the function f:

INPUT numbers $x_0, x_1, \ldots, x_n$; values $f(x_0), \ldots, f(x_n)$ and $f'(x_0), \ldots, f'(x_n)$.
OUTPUT the numbers $Q_{0,0}, Q_{1,1}, \ldots, Q_{2n+1,2n+1}$ where

$$H(x) = Q_{0,0} + Q_{1,1}(x - x_0) + Q_{2,2}(x - x_0)^2 + Q_{3,3}(x - x_0)^2(x - x_1) + Q_{4,4}(x - x_0)^2(x - x_1)^2 + \cdots + Q_{2n+1,2n+1}(x - x_0)^2(x - x_1)^2 \cdots (x - x_{n-1})^2(x - x_n).$$

Step 1 For $i = 0, 1, \ldots, n$ do Steps 2 and 3.

Step 2 Set $z_{2i} = x_i$;
$z_{2i+1} = x_i$;
$Q_{2i,0} = f(x_i)$;
$Q_{2i+1,0} = f(x_i)$;
$Q_{2i+1,1} = f'(x_i)$.

Step 3 If $i \neq 0$ then set

$$Q_{2i,1} = \frac{Q_{2i,0} - Q_{2i-1,0}}{z_{2i} - z_{2i-1}}.$$

Step 4 For $i = 2, 3, \ldots, 2n+1$

for $j = 2, 3, \ldots, i$ set $Q_{i,j} = \dfrac{Q_{i,j-1} - Q_{i-1,j-1}}{z_i - z_{i-j}}$.

Step 5 OUTPUT $(Q_{0,0}, Q_{1,1}, \ldots, Q_{2n+1,2n+1})$;
STOP

Algorithm 3.4: HERMITE INTERPOLATION

The YouTube video developed by NPTEL can serve as a good illustration of the Forward Divided Difference Formula.

▸ Hermite Interpolation Video

Definition (3.10)

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:

(a) $S(x)$ is a cubic polynomial, denoted $S_j(x)$, on the subinterval $[x_j, x_{j+1}]$ for each $j = 0, 1, \ldots, n-1$;

(b) $S_j(x_j) = f(x_j)$ and $S_j(x_{j+1}) = f(x_{j+1})$ for each $j = 0, 1, \ldots, n-1$;

(c) $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ for each $j = 0, 1, \ldots, n-2$; (*Implied by* **(b)**.)

(d) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ for each $j = 0, 1, \ldots, n-2$;

(e) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ for each $j = 0, 1, \ldots, n-2$;

(f) One of the following sets of boundary conditions is satisfied:

- **(i)** $S''(x_0) = S''(x_n) = 0$ **(natural** *(or free)* **boundary)**;
- **(ii)** $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ **(clamped boundary)**.

Theorem (3.11)

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 $x_0, x_1, \ldots, x_n$; that is, a spline interpolant that satisfies the natural boundary conditions $S''(a) = 0$ and $S''(b) = 0$.

Algorithm 3.4: NATURAL CUBIC SPLINE

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 $l_0 = 1$; $\mu_0 = 0$; $z_0 = 0$. (*Steps 3-5, & part of 6 solve tridiagonal linear system by method in Algorithm 6.7.*)

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 $c_j = z_j - \mu_j c_{j+1}$; $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$;
$b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$;

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

Chapter 3.5: Cubic Spline Interpolation

The YouTube video developed by Oscar Veliz can serve as a good illustration of the Cubic Spline. ▸ Cubic Spline Video

Theorem (3.12)

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 $x_0, x_1, \ldots, x_n$; that is, a spline interpolant that satisfies the clamped boundary conditions $S'(a) = f'(a)$ and $S'(b) = f'(b)$.

Algorithm 3.5: CLAMPED CUBIC SPLINE

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

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 $j = 0, 1, \ldots, n-1$.

(*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 Set $\alpha_0 = 3(a_1 - a_0)/h_0 - 3FPO$; $\alpha_n = 3FPN - 3(a_n - a_{n-1})/h_{n-1}$.
$\alpha_n = 3FPN - 3(a_n - a_{n-1})/h_{n-1}$.
Step 3 For $i = 1, 2, \ldots, n-1$

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

Step 4 Set $l_0 = 2h_0$; $\mu_0 = 0.5$; $_0 = \alpha_0/l_0$. (*Steps 4-6, & part of 7 solve a tridiagonal linear system by method in Algorithm 6.7.*)
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.

Theorem (3.13)

Let $f \in C^4[a,b]$ with $\max_{a\le x\le b}|f^{(4)}(x)| = M$. If S is the unique clamped cubic spline interpolant to f with respect to the nodes $a = x_0 < x_1 < \cdots < x_n = b$, then for all x in $[a,b]$,

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

Algorithm 3.6: BÉZIER CURVE

To construct the cubic Bézier curves $C_0, \ldots, C_{n-1}$ in parametric form, where C_i is represented by

$$(x_i(t), y_i(t)) = (a_0^{(i)} + a_1^{(i)}t + a_2^{(i)}t^2 + a_3^{(i)}t^3, b_0^{(i)} + b_1^{(i)}t + b_2^{(i)}t^2 + b_3^{(i)}t^3),$$

for $0 \le t \le 1$, as determined by the left endpoint (x_i, y_i), left guidepoint (x_i^+, y_i^+), right endpoint (x_{i+1}, y_{i+1}), and right guidepoint (x_{i+1}^-, y_{i+1}^-) for each $i = 0, 1, \ldots, n-1$:
INPUT n; $(x_0, y_0), \ldots, (x_n, y_n)$; $(x_0^+, y_0^+), \ldots, (x_{n-1}^+, y_{n-1}^+)$; $(x_1^-, y_1^-), \ldots, (x_n^-, y_n^-)$.
OUTPUT coefficients $\{a_0^{(i)}, a_1^{(i)}, a_2^{(i)}, a_3^{(i)}, b_0^{(i)}, b_1^{(i)}, b_2^{(i)}, b_3^{(i)}$, for $0 \le i \le n-1\}$.

Step 1 For each $i = 0, 1, \ldots, n-1$ do Steps 2 and 3.

Step 2 Set $a_0^{(i)} = x_i$; $b_0^{(i)} = y_i$; $a_1^{(i)} = 3(x_i^+ - x_i)$; $b_1^{(i)} = 3(y_i^+ - y_i)$;
$a_2^{(i)} = 3(x_i + x_{i+1}^- - 2x_i^+)$; $b_2^{(i)} = 3(y_i + y_{i+1}^- - 2y_i^+)$;
$a_3^{(i)} = x_{i+1} - x_i + 3x_i^+ - 3x_{i+1}^-$; $b_3^{(i)} = y_{i+1} - y_i + 3y_i^+ - 3y_{i+1}^-$;

Step 3 OUTPUT $(a_0^{(i)}, a_1^{(i)}, a_2^{(i)}, a_3^{(i)}, b_0^{(i)}, b_1^{(i)}, b_2^{(i)}, b_3^{(i)})$.

Step 4 STOP.

Chapter 3.6: Bézier Curve

The following website links to a Java applet that can serve as a good illustration of the Bézier Curve. This applet works best in Firefox. The URL may need to be added to the Java security exceptions. ▸ Bézier Curve Java Applet