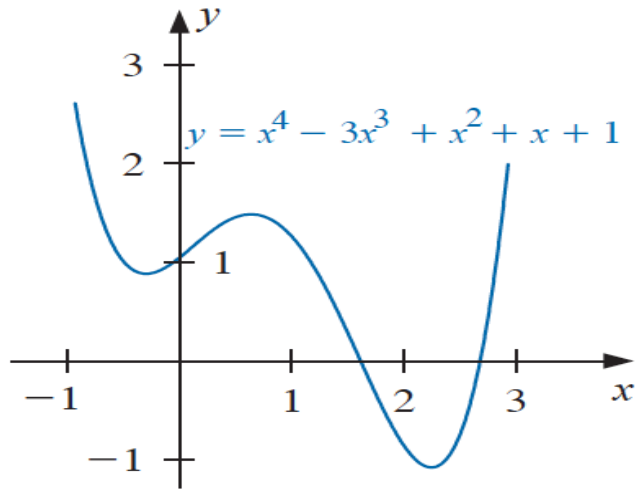
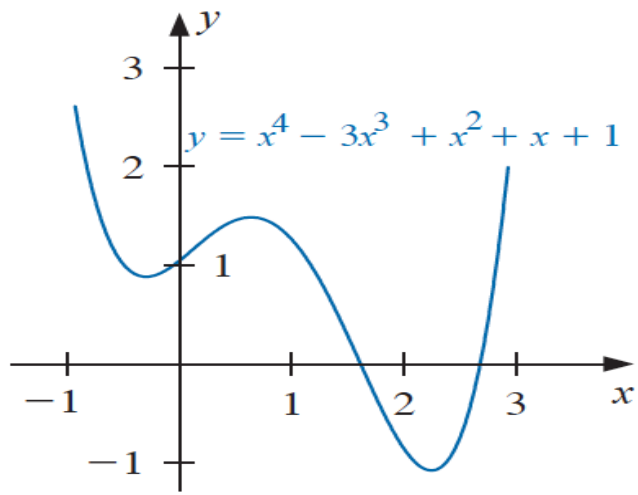
Muller's Method: $p_0 = 0.5, p_1 = -0.5, p_2 = 0$, root $p \approx -0.339 + 0.447i$

- Function has real & complex roots.

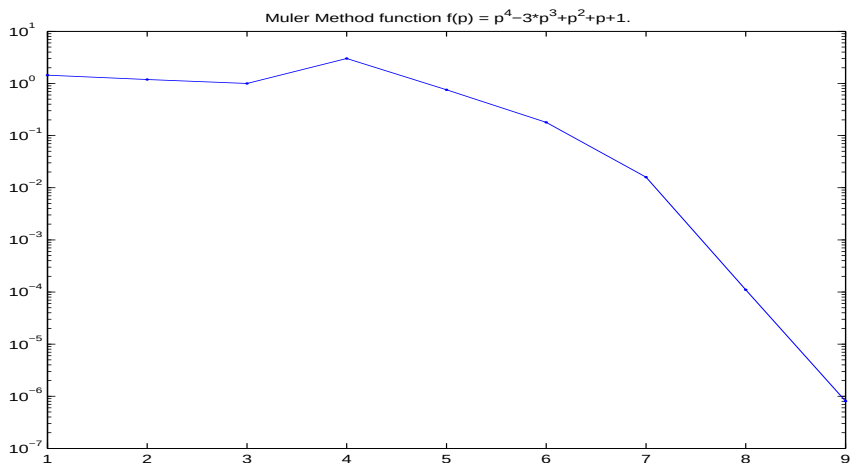


Muller's Method: $p_0 = 0.5, p_1 = -0.5, p_2 = 0$, root $p \approx -0.339 + 0.447i$

- Function has real & complex roots.



- Muller's method converges to complex root in 9 iterations.



Order of Convergence: Muller's Method

Given three approximations p_{n-2}, p_{n-1}, p_n, Muller's parabola

$$\begin{aligned} P(x) &= a(x-p_n)^2 + b(x-p_n) + c \quad \text{satisfies} \\ f(p_j) &= P(p_j), \quad j = n-2, n-1, n, \quad \text{and} \quad P(p_{n+1}) = 0. \end{aligned}$$

Assume that

- Muller's converges to root p.
- $f'''(x)$ is continuous with $f'(p) \neq 0$.

Order of Convergence: Muller's Method

Given three approximations p_{n-2}, p_{n-1}, p_n, Muller's parabola

$$\begin{aligned} P(x) &= a(x-p_n)^2 + b(x-p_n) + c \quad \text{satisfies} \\ f(p_j) &= P(p_j), \quad j = n-2, n-1, n, \quad \text{and} \quad P(p_{n+1}) = 0. \end{aligned}$$

Assume that

- Muller's converges to root p.
- $f'''(x)$ is continuous with $f'(p) \neq 0$.

Theorem: Muller's Method converges at order $\mu \approx 1.84$, with $\mu^3 = \mu^2 + \mu + 1$.

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^\mu} = \texttt{constant}.$$

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n x^n + b_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find all n roots in $P(x) = 0$.

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n\,x^n + b_{n-1}\,x^{n-1} + \cdots + a_2\,x^2 + a_1\,x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find <u>all</u> n roots in $P(x) = 0$.

(2) Let Horner's Method compute

$$P(x) = (x - x_0)\,Q(x) + b_0, \text{ with } Q(x) = b_n\,x^{n-1} + b_{n-1}\,x^{n-2} + \cdots + b_2\,x + b_1,$$

after finding x_0 with a root-finder (*i.e.* Newton, bisection, or Muller.)

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n\,x^n + b_{n-1}\,x^{n-1} + \cdots + a_2\,x^2 + a_1\,x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find all n roots in $P(x) = 0$.

(2) Let Horner's Method compute

$$P(x) = (x - x_0)\,Q(x) + b_0, \text{ with } Q(x) = b_n\,x^{n-1} + b_{n-1}\,x^{n-2} + \cdots + b_2\,x + b_1,$$

after finding x_0 with a root-finder (*i.e.* Newton, bisection, or Muller.)

(3) An approximate root $x \approx x_0$ of $P(x) = 0$ is found if

$$|b_0| \leq \texttt{tol} \quad \text{for tolerance} \quad \texttt{tol}. \quad (1)$$

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n\,x^n + b_{n-1}\,x^{n-1} + \cdots + a_2\,x^2 + a_1\,x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find all n roots in $P(x) = 0$.

(2) Let Horner's Method compute

$$P(x) = (x - x_0)\,Q(x) + b_0, \text{ with } Q(x) = b_n\,x^{n-1} + b_{n-1}\,x^{n-2} + \cdots + b_2\,x + b_1,$$

after finding x_0 with a root-finder (*i.e.* Newton, bisection, or Muller.)

(3) An approximate root $x \approx x_0$ of $P(x) = 0$ is found if

$|b_0| \leq \texttt{tol}$ for tolerance $\texttt{tol}$. (1) $\Longrightarrow P(x) \approx (x - x_0)\,Q(x)$

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n\,x^n + b_{n-1}\,x^{n-1} + \cdots + a_2\,x^2 + a_1\,x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find <u>all</u> n roots in $P(x) = 0$.

(2) Let Horner's Method compute

$$P(x) = (x - x_0)\,Q(x) + b_0, \text{ with } Q(x) = b_n\,x^{n-1} + b_{n-1}\,x^{n-2} + \cdots + b_2\,x + b_1,$$

after finding x_0 with a root-finder (*i.e.* Newton, bisection, or Muller.)

(3) An approximate root $x \approx x_0$ of $P(x) = 0$ is found if

$|b_0| \leq \texttt{tol}$ for tolerance $\texttt{tol}$. (1) $\Longrightarrow P(x) \approx (x - x_0)\,Q(x)$

(4) Find all other $n - 1$ roots in $P(x) = 0$ by recursively solving

$$Q(x) = 0.$$

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n\,x^n + b_{n-1}\,x^{n-1} + \cdots + a_2\,x^2 + a_1\,x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find <u>all</u> n roots in $P(x) = 0$.

(2) Let Horner's Method compute

$$P(x) = (x - x_0)\,Q(x) + b_0, \text{ with } Q(x) = b_n\,x^{n-1} + b_{n-1}\,x^{n-2} + \cdots + b_2\,x + b_1,$$

after finding x_0 with a root-finder (*i.e.* Newton, bisection, or Muller.)

(3) An approximate root $x \approx x_0$ of $P(x) = 0$ is found if

$|b_0| \leq \texttt{tol}$ for tolerance $\texttt{tol}$. (1) $\Longrightarrow P(x) \approx (x - x_0)\,Q(x)$

(4) Find all other $n - 1$ roots in $P(x) = 0$ by recursively solving

$$Q(x) = 0.$$

QUESTION: Can we find all polynomial roots now?

Deflation Procedure for finding all roots of a Polynomial

(1) Given polynomial $P(x)$ of degree n

$$P(x) = a_n\,x^n + b_{n-1}\,x^{n-1} + \cdots + a_2\,x^2 + a_1\,x + a_0 \quad \text{with} \quad a_n \neq 0.$$

Would like to find <u>all</u> n roots in $P(x) = 0$.

(2) Let Horner's Method compute

$$P(x) = (x - x_0)\,Q(x) + b_0, \text{ with } Q(x) = b_n\,x^{n-1} + b_{n-1}\,x^{n-2} + \cdots + b_2\,x + b_1,$$

after finding x_0 with a root-finder (*i.e.* Newton, bisection, or Muller.)

(3) An approximate root $x \approx x_0$ of $P(x) = 0$ is found if

$|b_0| \leq \texttt{tol}$ for tolerance $\texttt{tol}$. (1) $\Longrightarrow P(x) \approx (x - x_0)\,Q(x)$

(4) Find all other $n - 1$ roots in $P(x) = 0$ by recursively solving

$$Q(x) = 0.$$

QUESTION: Can we find all polynomial roots now?

Well $\cdots$

Stressed by choice and selection

Bisection? Fixed Point Iteration? Newton's Method?

Secant Method? Steffensen's Method? Muller's Method?

Choice and selection

- Bisection Method:
  - slow (linearly convergent);
  - always works for given interval $[a, b]$ with $f(a) \cdot f(b) < 0$.
- Fixed Point Iteration:
  - slow (linearly convergent);
  - need not work.
- Newton's Method:
  - fast (quadratically convergent);
  - needs derivatives; could get burnt (need not converge.)
- Secant Method:
  - between Bisection and Newton in speed;
  - need not converge.
- Steffensen's Method:
  - faster than Secant (quadratically convergent);
  - hard to stop; need not converge.
- Muller's Method:
  - handles complex roots;
  - need not converge.

Brent's Method (*a.k.a.* zeroin): Motivation

- Does not need derivatives.
- (Mostly) speed of
  Muller's Method
- (Mostly) reliability of
  Bisection Method
- Real roots only

Brent's Method (*a.k.a.* zeroin): Motivation

- Does not need derivatives.
- (Mostly) speed of
  Muller's Method
- (Mostly) reliability of
  Bisection Method
- Real roots only

Zero In Japanese cover

Brent's Method (*a.k.a.* zeroin): Motivation

- Does not need derivatives.
- (Mostly) speed of
  Muller's Method
- (Mostly) reliability of
  Bisection Method
- Real roots only

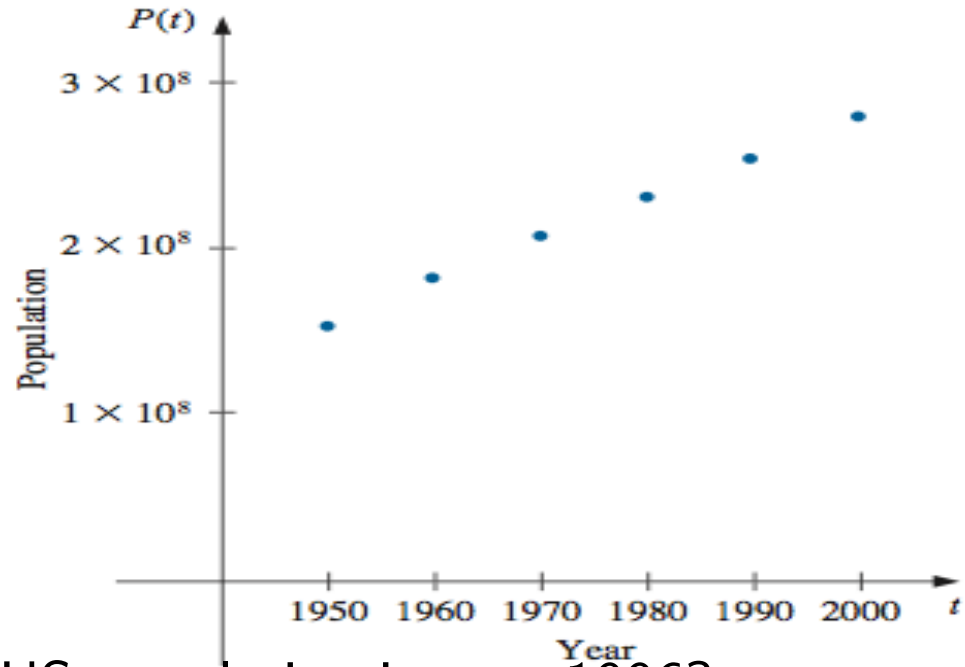
Zero In Japanese cover

A variant of Brent's Method will be programming project.

§3.1 Interpolation/Approximation (connecting the dots)

US population census data available every ten years.

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



- What was US population in year 1996? (INTERPOLATION)
- What would US population be in year 2017? (EXTRAPOLATION, typically more dangerous)

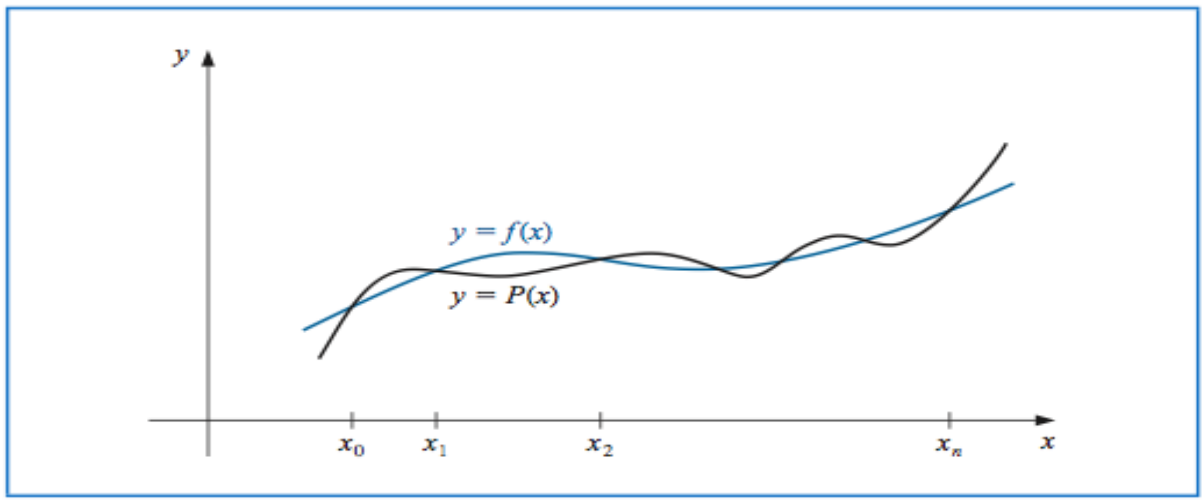
Connecting $n+1$ dots

- Given $n+1$ distinct points

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

- Find a degree $\leq n$ polynomial

$$\begin{aligned} P(x) &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \\ \text{so that } P(x_0) &= f(x_0), P(x_1) = f(x_1), \cdots, P(x_n) = f(x_n). \end{aligned}$$



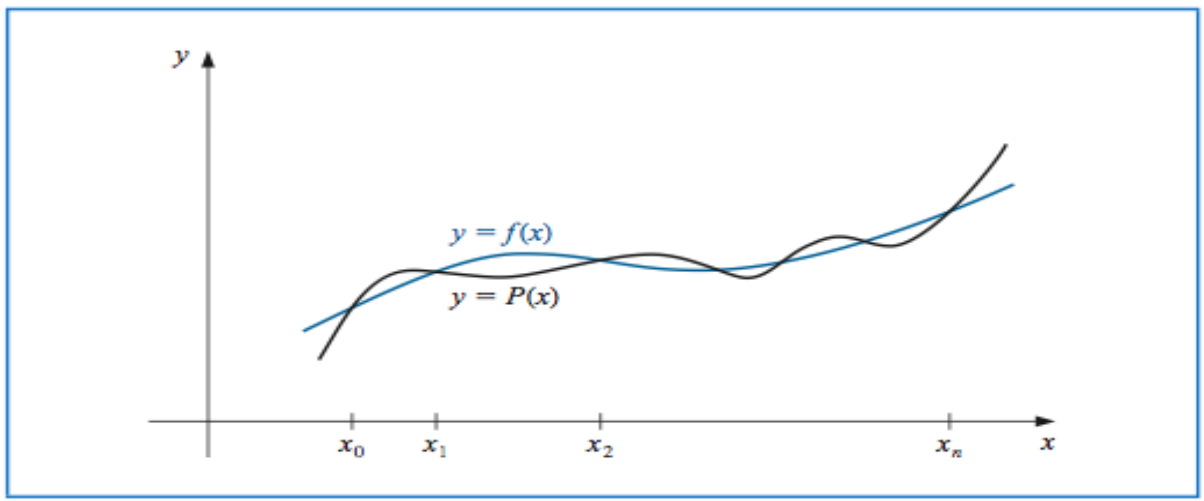
Connecting $n+1$ dots

- Given $n+1$ distinct points

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

- Find a degree $\leq n$ polynomial

$$\begin{aligned} P(x) &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \\ \text{so that } P(x_0) &= f(x_0), P(x_1) = f(x_1), \cdots, P(x_n) = f(x_n). \end{aligned}$$



https://medium.com/predictive-analytics-for-stock-market-prediction/machine-learning-algorithm-for-stock-prediction-e75c678d05c7

Connecting $n+1$ dots

- Would like to write $P(x)$ as

$$P(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x),$$

where $L_0(x), L_1(x), \cdots, L_n(x)$ are polynomials
UNRELATED
to $f(x_0), f(x_1), \cdots, f(x_n)$.

- at each node x_j, $j = 0, 1, \cdots, n$,

$$\begin{aligned} f(x_j) &= P(x_j) = f(x_0)L_0(x_j) + f(x_1)L_1(x_j) + \cdots + f(x_n)L_n(x_j) \\ &= 0 \cdot L_0(x_j) + \cdots + f(x_j) \cdot L_j(x_j) + \cdots + 0 \cdot L_n(x_j) \end{aligned}$$

This suggests for all i, j,

$$L_i(x_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases}$$

Connecting $n+1$ dots

- Would like to write $P(x)$ as

$$P(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x),$$

where $L_0(x), L_1(x), \cdots, L_n(x)$ are polynomials
UNRELATED
to $f(x_0), f(x_1), \cdots, f(x_n)$.

- at each node x_j, $j = 0, 1, \cdots, n$,

$$\begin{aligned} f(x_j) &= P(x_j) = f(x_0)L_0(x_j) + f(x_1)L_1(x_j) + \cdots + f(x_n)L_n(x_j) \\ &= 0 \cdot L_0(x_j) + \cdots + f(x_j) \cdot L_j(x_j) + \cdots + 0 \cdot L_n(x_j) \end{aligned}$$

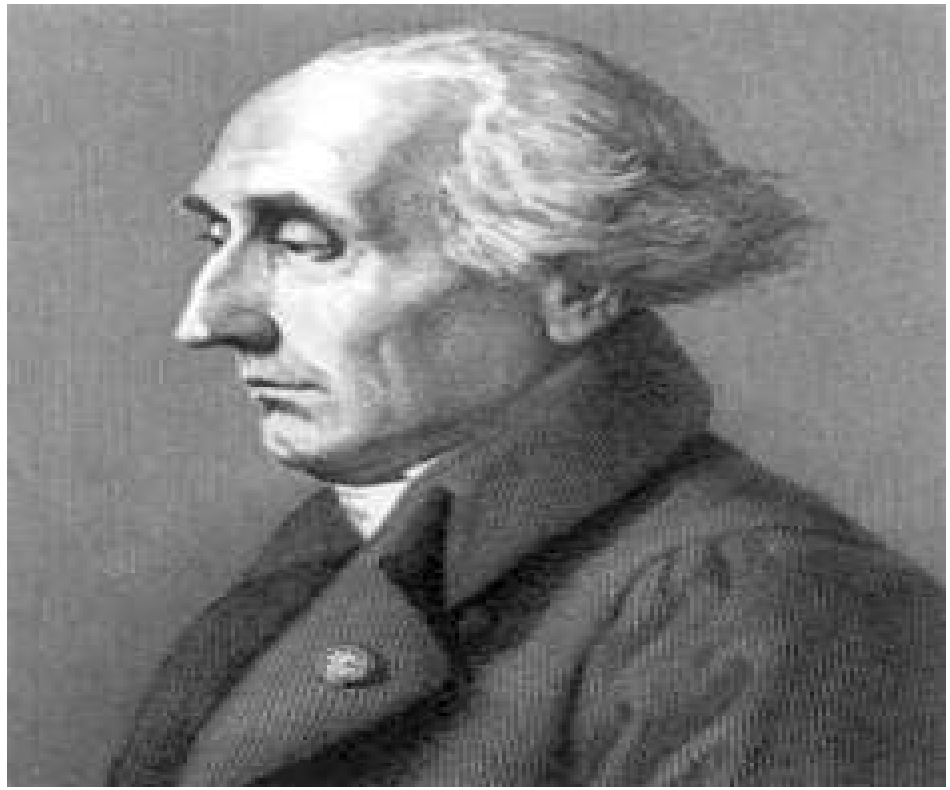
This suggests for all i, j,

$$L_i(x_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases}$$

So

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

All things Lagrangian (1736 – 1813)



Lagrange spectrum	Lagrangian coordinates	Lagrange crater
Lagrangian system	Lagrangian subspace	Lagrangian point
Lagrange resolvent	Lagrangian mechanics	Lagrange polynomial
Lagrange's theorem	Lagrangian analysis	Lagrange multiplier

Quadratic Interpolation

- Use nodes $x_0 = 2$, $x_1 = 2.75$, $x_2 = 4$ to find the second Lagrange interpolating polynomial for $f(x) = \frac{1}{x}$
- Use this polynomial to approximate $f(3) = \frac{1}{3}$

Quadratic Interpolation

- Use nodes $x_0 = 2$, $x_1 = 2.75$, $x_2 = 4$ to find the second Lagrange interpolating polynomial for $f(x) = \frac{1}{x}$
- Use this polynomial to approximate $f(3) = \frac{1}{3}$

$$L_0(x) = \frac{(x-2.75)(x-4)}{(2-2.5)(2-4)} = \frac{2}{3}(x-2.75)(x-4),$$

$$L_1(x) = \frac{(x-2)(x-4)}{(2.75-2)(2.75-4)} = -\frac{16}{15}(x-2)(x-4),$$

$$L_2(x) = \frac{(x-2)(x-2.75)}{(4-2)(4-2.5)} = \frac{2}{5}(x-2)(x-2.75).$$

Quadratic Interpolation

- Use nodes $x_0 = 2$, $x_1 = 2.75$, $x_2 = 4$ to find the second Lagrange interpolating polynomial for $f(x) = \frac{1}{x}$
- Use this polynomial to approximate $f(3) = \frac{1}{3}$

$$L_0(x) = \frac{(x-2.75)(x-4)}{(2-2.5)(2-4)} = \frac{2}{3}(x-2.75)(x-4),$$

$$L_1(x) = \frac{(x-2)(x-4)}{(2.75-2)(2.75-4)} = -\frac{16}{15}(x-2)(x-4),$$

$$L_2(x) = \frac{(x-2)(x-2.75)}{(4-2)(4-2.5)} = \frac{2}{5}(x-2)(x-2.75).$$

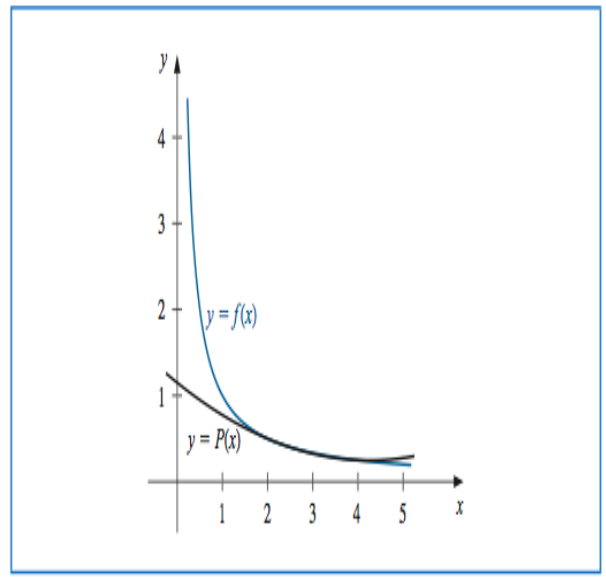
$$\begin{aligned} P(x) &= \sum_{k=0}^{2} f(x_k)L_k(x) \\ &= \frac{1}{3}(x-2.75)(x-4) - \frac{64}{165}(x-2)(x-4) + \frac{1}{10}(x-2)(x-2.75) \\ &= \frac{1}{22}x^2 - \frac{35}{88}x + \frac{49}{44}. \end{aligned}$$

Quadratic Interpolation: Example

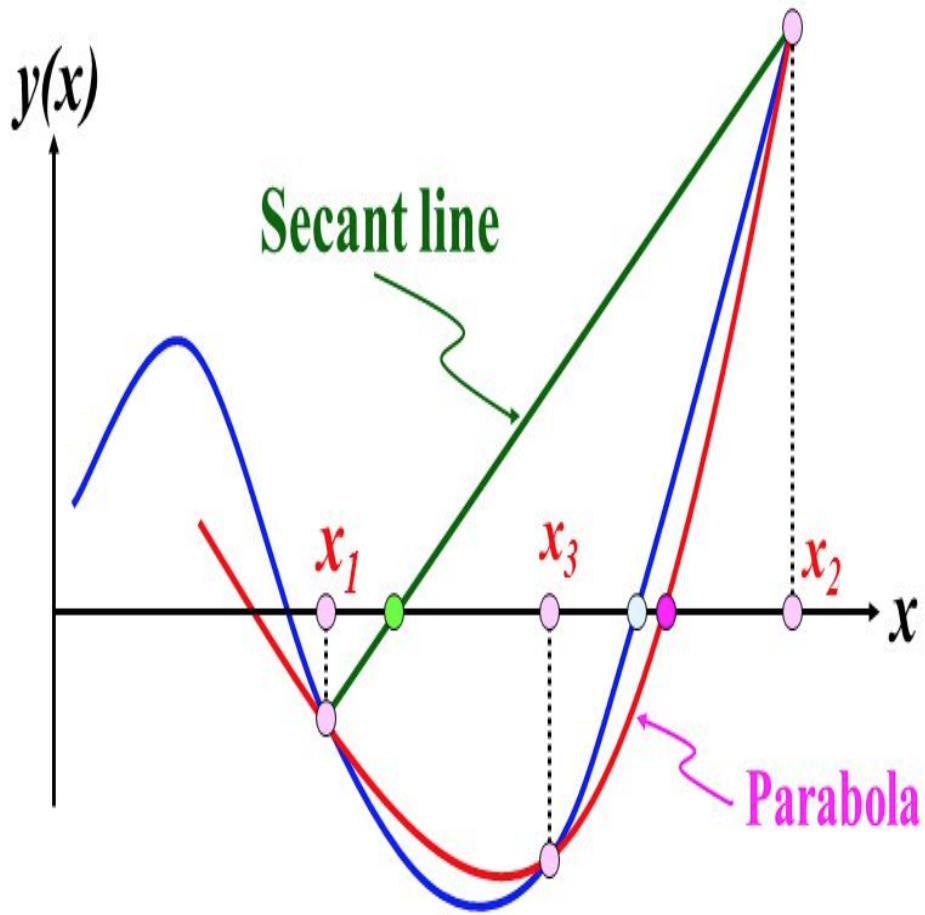
$$f(3) \approx P(3) = \frac{9}{22} - \frac{105}{88} + \frac{49}{44} \approx 0.32955$$

Quadratic Interpolation: Example

$$f(3) \approx P(3) = \frac{9}{22} - \frac{105}{88} + \frac{49}{44} \approx 0.32955$$



Applications: Secant Method = linear interpolation
Muller's Method = quadratic interpolation



- Secant Method requires two points $(x_1, f(x_1)), (x_2, f(x_2))$.
- Muller's Method requires three points $(x_1, f(x_1)), (x_2, f(x_2))$, and $(x_3, f(x_3))$.

Polynomial Interpolation with $n+1$ nodes

- Given $n+1$ distinct points

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

- Interpolating polynomial of degree $\leq n$

$$\begin{aligned} P(x) &= f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x), \\ \text{with } P(x_0) &= f(x_0), \quad P(x_1) = f(x_1), \quad \cdots, \quad P(x_n) = f(x_n), \end{aligned}$$

and Lagrangian polynomials

$$L_i(x_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases} \qquad L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

Polynomial Interpolation with $n+1$ nodes

- Given $n+1$ distinct points

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

- Interpolating polynomial of degree $\leq n$

$$\begin{aligned} P(x) &= f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x), \\ \text{with } P(x_0) &= f(x_0), \quad P(x_1) = f(x_1), \quad \cdots, \quad P(x_n) = f(x_n), \end{aligned}$$

and Lagrangian polynomials

$$L_i(x_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases} \qquad L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

- **Analysis**: Estimate errors with polynomial interpolation
- **Computation**: How to compute $P(x)$ numerically

Polynomial Interpolation: Analysis

Lemma: Let $f(x) \in C^n[a, b]$ be such that

$$f(x_1) = 0, \quad f(x_2) = 0, \quad \cdots, \quad f(x_n) = 0,$$

where $a \leq x_1 < x_2 < \cdots < x_n \leq b$ are mutually distinct. Then there exists a $\xi \in [a, b]$ such that

$$f^{(n-1)}(\xi) = 0. \qquad (1)$$

Polynomial Interpolation: Analysis

Lemma: Let $f(x) \in C^n[a, b]$ be such that

$$f(x_1) = 0, \quad f(x_2) = 0, \quad \cdots, \quad f(x_n) = 0,$$

where $a \leq x_1 < x_2 < \cdots < x_n \leq b$ are mutually distinct. Then there exists a $\xi \in [a, b]$ such that

$$f^{(n-1)}(\xi) = 0. \qquad (1)$$

Proof:

- There must exist $x_i < y_i < x_{i+1}$ so that

$$f^{(1)}(y_i) = 0, \quad \text{for} \quad i = 1, \cdots, n-1.$$

Polynomial Interpolation: Analysis

Lemma: Let $f(x) \in C^n[a, b]$ be such that

$$f(x_1) = 0, \quad f(x_2) = 0, \quad \cdots, \quad f(x_n) = 0,$$

where $a \leq x_1 < x_2 < \cdots < x_n \leq b$ are mutually distinct. Then there exists a $\xi \in [a, b]$ such that

$$f^{(n-1)}(\xi) = 0. \qquad (1)$$

Proof:

- There must exist $x_i < y_i < x_{i+1}$ so that

$$f^{(\underline{1})}(y_i) = 0, \quad \text{for} \quad i = 1, \cdots, \underline{n-1}.$$

- There must exist $y_i < z_i < y_{i+1}$ so that

$$f^{(\underline{2})}(z_i) = 0, \quad \text{for} \quad i = 1, \cdots, \underline{n-2}.$$

Polynomial Interpolation: Analysis

Lemma: Let $f(x) \in C^n[a, b]$ be such that

$$f(x_1) = 0, \quad f(x_2) = 0, \quad \cdots, \quad f(x_n) = 0,$$

where $a \leq x_1 < x_2 < \cdots < x_n \leq b$ are mutually distinct. Then there exists a $\xi \in [a, b]$ such that

$$f^{(n-1)}(\xi) = 0. \qquad (1)$$

Proof:

- There must exist $x_i < y_i < x_{i+1}$ so that

$$f^{(\underline{1})}(y_i) = 0, \quad \text{for} \quad i = 1, \cdots, \underline{n-1}.$$

- There must exist $y_i < z_i < y_{i+1}$ so that

$$f^{(\underline{2})}(z_i) = 0, \quad \text{for} \quad i = 1, \cdots, \underline{n-2}.$$

- Repeat $n - 1$ times to reach (1).

Polynomial 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 $x \in [a, b]$, a number $\xi(x)$ between $, x_0, x_1, \cdots, x_n$ (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.

Polynomial Interpolation Error: Proof

If $x = x_0, x_1, \cdots, \ x_n$, then error $= 0$ and theorem is true. Now let x be not equal to any node. Define function g for $t \in [a, b]$

$$\begin{aligned} g(t) &\overset{def}{=} (f(t) - P(t)) - (f(x) - P(x)) \frac{(t - x_0)(t - x_1) \cdots (t - x_n)}{(x - x_0)(x - x_1) \cdots (x - x_n)} \\ &= (f(t) - P(t)) - (f(x) - P(x)) \prod_{j=0}^{n} \frac{(t - x_j)}{(x - x_j)} \in C^{n+1}[a, b]. \end{aligned}$$

Then $g(t)$ vanishes at $n + 2$ distinct points:

$$g(x) = 0, \quad g(x_k) = 0, \quad \text{, for} \quad k = 0, 1, \cdots, n.$$

There must be a ξ between x and nodal points such that

$$g^{(n+1)}(\xi) = 0.$$

Polynomial Interpolation Error: Proof

Since

$$g^{(n+1)}(\xi) = (f(t) - P(t))^{(n+1)}\,|_{t=\xi} - (f(x) - P(x))\Big(\prod_{j=0}^{n} \frac{(t - x_j)}{(x - x_j)}\Big)^{(n+1)}\,|_{t=\xi}$$

$$= f^{(n+1)}(\xi) - (f(x) - P(x))\,\frac{(n+1)!}{\prod_{j=0}^{n}(x - x_j)}$$

$$= 0$$

Therefore

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

Corollary: Suppose $x_0, x_1, \cdots, x_n$ are distinct numbers in the interval $[a, b]$ and f is polynomial of degree at most n, then

$$P(x) = f(x).$$

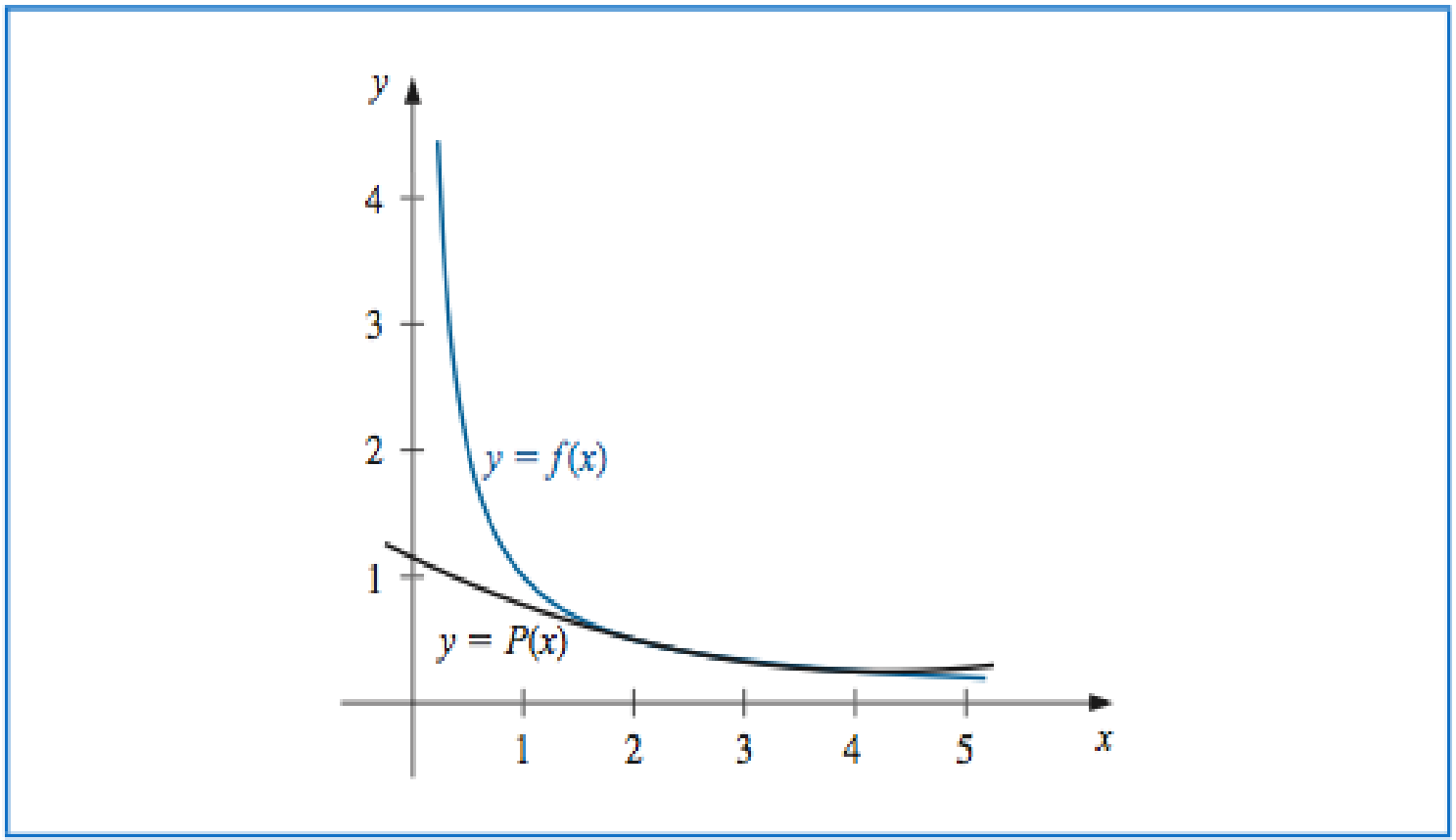
Corollary: Suppose $x_0, x_1, \cdots, x_n$ are distinct numbers in the interval $[a, b]$ and f is polynomial of degree at most n, then

$$P(x) = f(x).$$

Theorem: Assume that the nodal points $x_0, \cdots, x_n$ are mutually distinct, then the interpolating polynomial $P(x)$ of degree $\leq n$ is unique.

Polynomial Interpolation, Error Bounds:

Estimate maximum error in second order polynomial interpolation of function $f(x) = \frac{1}{x}$ over interval $[2, 4]$ using nodes $x_0 = 2, x_1 = 2.75$, and $x_2 = 4$.



Polynomial Interpolation, Error Bounds

$$f(x) = P(x) + \frac{f^{(3)}(\xi(x))}{3!}(x-2)(x-2.75)(x-4),$$

where $P(x)$ is the interpolating polynomial on $[2, 4]$.

- Find upper bounds on

  $$\left|f^{(3)}(\xi(x))\right| \quad \text{and} \quad |(x-2)(x-2.75)(x-4)| \quad \text{for} \quad x \in [2,4]$$

  separately, then put them together

Polynomial Interpolation, Error Bounds

$$f(x) = P(x) + \frac{f^{(3)}(\xi(x))}{3!}(x-2)(x-2.75)(x-4),$$

where $P(x)$ is the interpolating polynomial on $[2,4]$.

- Find upper bounds on

  $$\left|f^{(3)}(\xi(x))\right| \quad \text{and} \quad |(x-2)(x-2.75)(x-4)| \quad \text{for} \quad x \in [2,4]$$

  separately, then put them together

  $$f^{(3)}(x) \quad = \quad -6x^{-4}, \quad |f^{(3)}(\xi(x))| \leq 6 \cdot 2^{-4},$$

Polynomial Interpolation, Error Bounds

$$f(x) = P(x) + \frac{f^{(3)}(\xi(x))}{3!}(x-2)(x-2.75)(x-4),$$

where $P(x)$ is the interpolating polynomial on $[2,4]$.

- Find upper bounds on

  $$\left|f^{(3)}(\xi(x))\right| \quad \text{and} \quad |(x-2)(x-2.75)(x-4)| \quad \text{for} \quad x \in [2,4]$$

  separately, then put them together

$$\begin{aligned}
f^{(3)}(x) &= -6x^{-4}, \quad |f^{(3)}(\xi(x))| \leq 6 \cdot 2^{-4}, \\
g(x) &\stackrel{def}{=} (x-2)(x-2.75)(x-4) = x^3 - \frac{35}{4}x^2 + \frac{49}{2}x - 22. \\
\frac{d\,g(x)}{d\,x} &= 3x^2 - \frac{35}{2}x + \frac{49}{2} = \frac{1}{2}(3x-7)(2x-7). \\
|g(\frac{7}{2})| &= \frac{9}{16} = \mathbf{max}_{x\in[2,4]}|g(x)|. \quad (|g(\frac{7}{3})| = \frac{25}{108} < \frac{9}{16})
\end{aligned}$$

Polynomial Interpolation, Error Bounds

$$f(x) = P(x) + \frac{f^{(3)}(\xi(x))}{3!}(x-2)(x-2.75)(x-4),$$

where $P(x)$ is the interpolating polynomial on $[2,4]$.

- Find upper bounds on

  $$\left|f^{(3)}(\xi(x))\right| \quad \text{and} \quad |(x-2)(x-2.75)(x-4)| \quad \text{for} \quad x \in [2,4]$$

  separately, then put them together

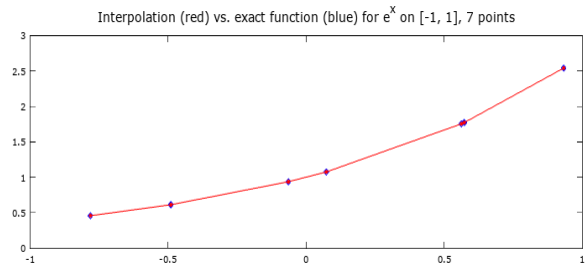
$$\begin{aligned}
f^{(3)}(x) &= -6x^{-4}, \quad |f^{(3)}(\xi(x))| \le 6 \cdot 2^{-4}, \\
g(x) &\stackrel{def}{=} (x-2)(x-2.75)(x-4) = x^3 - \frac{35}{4}x^2 + \frac{49}{2}x - 22. \\
\frac{d\,g(x)}{d\,x} &= 3x^2 - \frac{35}{2}x + \frac{49}{2} = \frac{1}{2}(3x-7)(2x-7). \\
|g(\tfrac{7}{2})| &= \frac{9}{16} = \mathbf{max}_{x\in[2,4]}|g(x)|. \quad (|g(\tfrac{7}{3})| = \frac{25}{108} < \frac{9}{16}) \\
|f(x) - P(x)| &\le \frac{1}{3!} \cdot 6 \cdot 2^{-4} \cdot \frac{9}{16} = \frac{9}{256}.
\end{aligned}$$

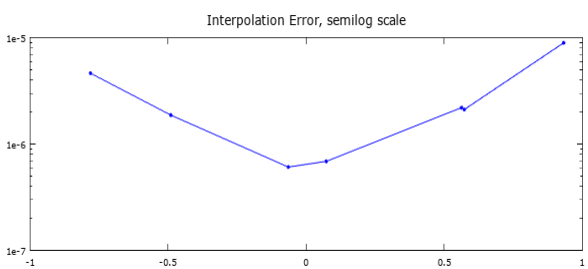
Polynomial Interpolation: Experiments:

- **Easy function**: $f(x) = e^x$ on $[-1, 1]$.

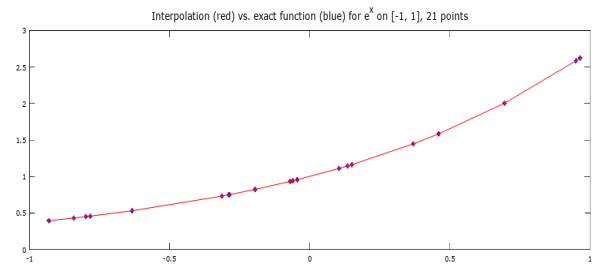
$$|f^{(n)}(\xi)| \leq e \quad \text{for all} \quad \xi \in (-1, 1).$$

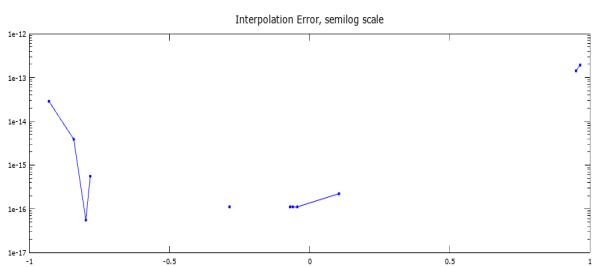
7 nodal points ($n = 6$) random x points.





21 nodal points ($n = 20$) random x points.



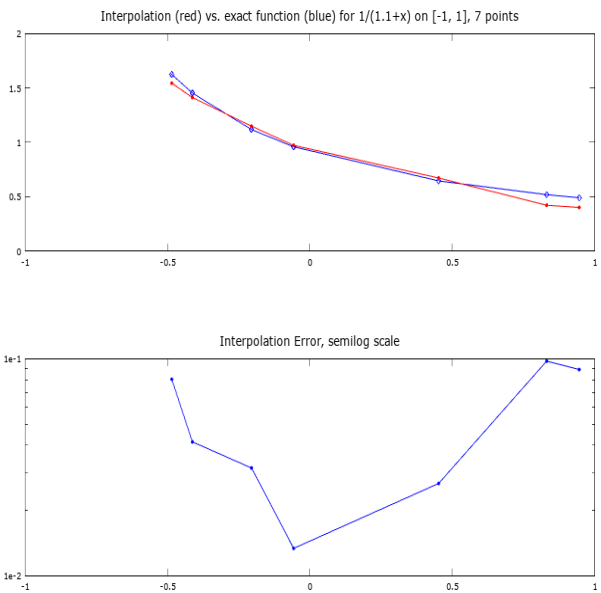


Polynomial Interpolation: Experiments:

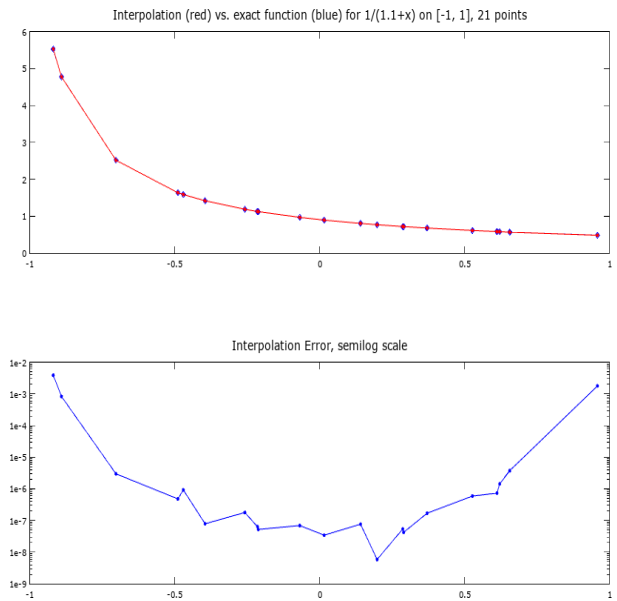
- **Hard function**: $f(x) = \frac{1}{1.1+x}$ on $[-1, 1]$.

$$|f^{(n)}(\xi)| = \frac{n!}{(1.1+\xi)^{n+1}} \leq 10^{n+1} \cdot n! \quad \text{for all} \quad \xi \in (-1, 1).$$

7 nodal points ($n = 6$) random x points.



21 nodal points ($n = 20$) random x points.



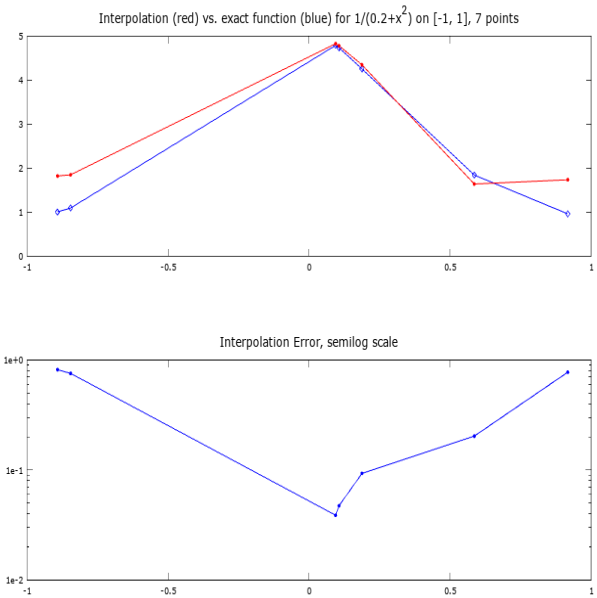
Polynomial Interpolation: Experiments:

- **Hardest function**: $f(x) = \frac{1}{0.2+x^2}$ on $[-1, 1]$.

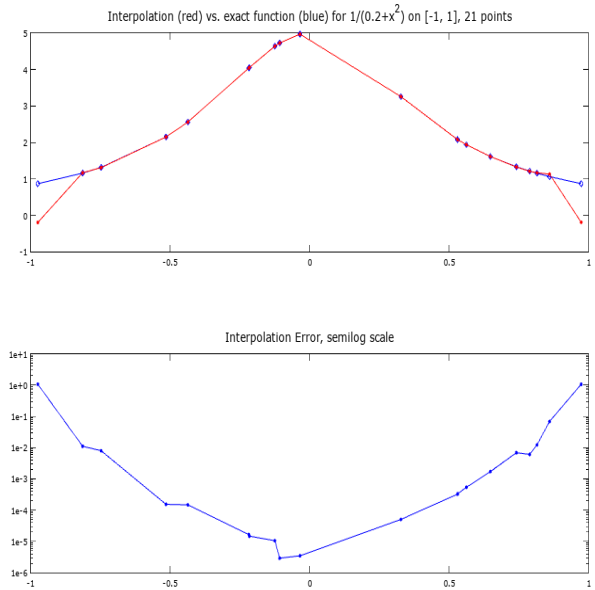
$$|f^{(n)}(\xi)| \quad \text{can be very large for some} \quad \xi \in (-1, 1).$$

- 7 nodal points ($n = 6$); random x points.

7 nodal points ($n = 6$)
random x points.

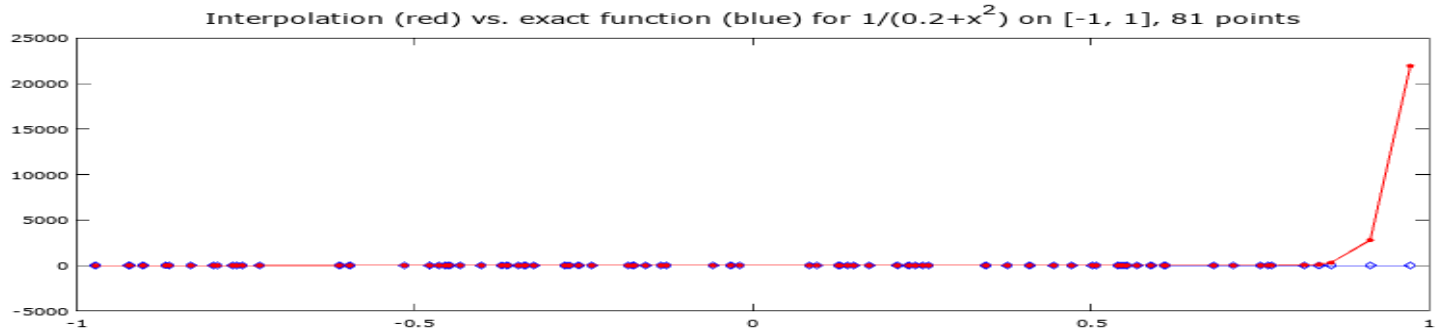


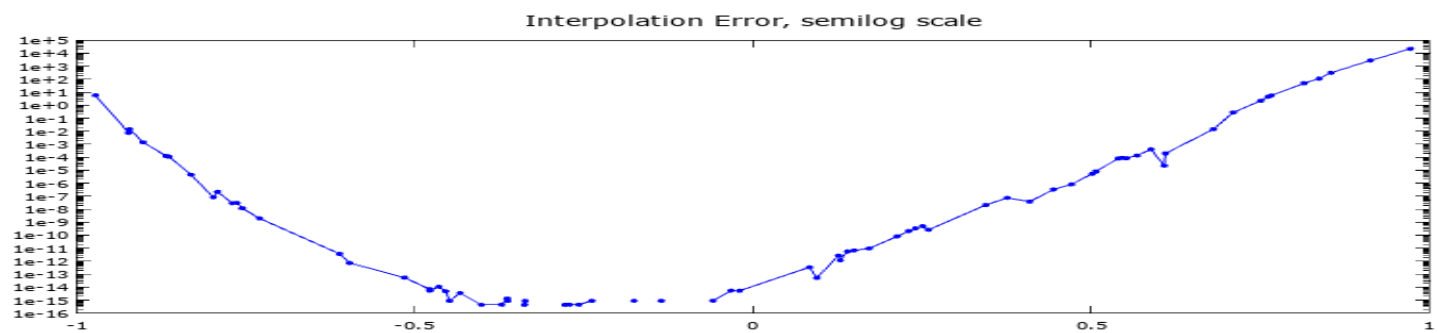
21 nodal points ($n = 20$)
random x points.



Polynomial Interpolation: Experiments:

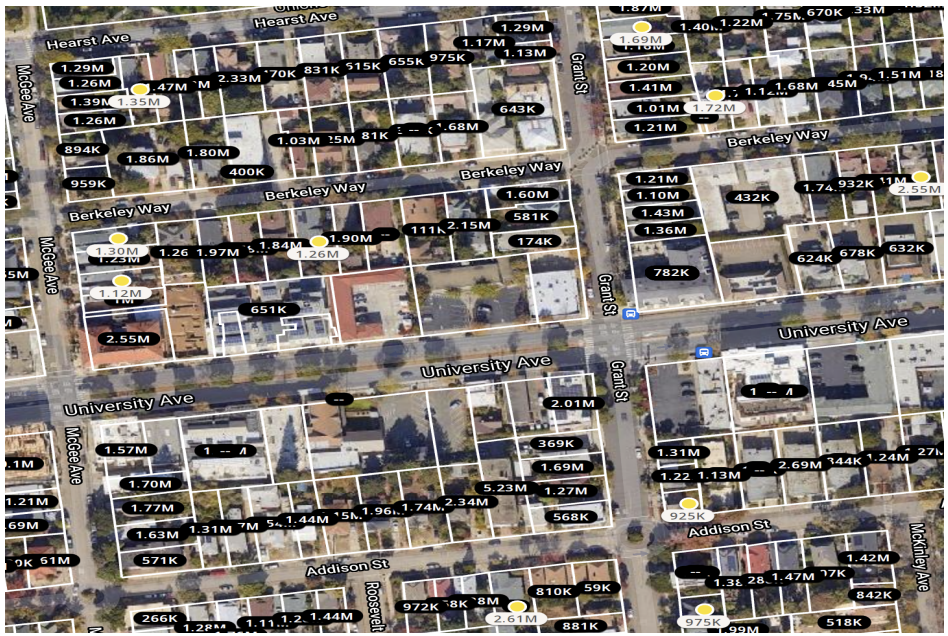
- **Hardest function**: $f(x) = \frac{1}{0.2+x^2}$ on $[-1, 1]$.
- 81 nodal points ($n = 80$); random x points.
- The harder you work, the worse you get.





Application: Home price predictions, based on recent sales

- Prediction as function of house size
- Prediction as function of lot size
- Prediction as function of location



It takes more than polynomial interpolation

Polynomial Interpolation: Analysis and Computation

- **Analysis**: Estimate errors with polynomial interpolation
- **Computation**: How to compute $P(x)$ numerically

§3.2 Polynomial Interpolation: Neville's Method (I)

- **Let** $Q(x)$ interpolate $f(x)$ at x_0, x_1,
- **Let** $\widehat{Q}(x)$ interpolate $f(x)$ at x_1, x_2.
- **Then**

$$P(x) \stackrel{def}{=} \frac{(x - x_2)Q(x) - (x - x_0)\widehat{Q}(x)}{x_0 - x_2}$$

interpolates $f(x)$ at x_0, x_1, x_2

Build higher degree interpolating polynomial from lower degree polys.

Polynomial Interpolation: Neville's Method (II)

- **Let** $Q(x)$ interpolate $f(x)$ at $x_0, x_1, \cdots, x_k$,
- **Let** $\widehat{Q}(x)$ interpolate $f(x)$ at $x_1, \cdots, x_k, x_{k+1}$.
- **Then**

$$P(x) \stackrel{def}{=} \frac{(x - x_{k+1})Q(x) - (x - x_0)\widehat{Q}(x)}{x_0 - x_{k+1}}$$

interpolates $f(x)$ at $x_0, x_1, \cdots, x_k, x_{k+1}$

Build higher degree interpolating polynomial from lower degree polys.

Neville's Method: Proof

- **Let** $Q(x)$ interpolate $f(x)$ at $x_0, x_1, \cdots, x_k$,
- **Let** $\widehat{Q}(x)$ interpolate $f(x)$ at $x_1, \cdots, x_k, x_{k+1}$.

$$P(x) \stackrel{def}{=} \frac{(x - x_{k+1})Q(x) - (x - x_0)\widehat{Q}(x)}{x_0 - x_{k+1}}$$

- **for** $1 \leq j \leq k$,

$$\begin{aligned} P(x_j) &= \frac{(x_j - x_{k+1})Q(x_j) - (x_j - x_0)\widehat{Q}(x_j)}{x_0 - x_{k+1}} \\ &= \frac{(x_j - x_{k+1})f(x_j) - (x_j - x_0)f(x_j)}{x_0 - x_{k+1}} = f(x_j). \end{aligned}$$

Neville's Method: Proof

- **Let** $Q(x)$ interpolate $f(x)$ at $x_0, x_1, \cdots, x_k$,
- **Let** $\widehat{Q}(x)$ interpolate $f(x)$ at $x_1, \cdots, x_k, x_{k+1}$.

$$P(x) \stackrel{def}{=} \frac{(x - x_{k+1})Q(x) - (x - x_0)\widehat{Q}(x)}{x_0 - x_{k+1}}$$

- **for** $j = 0, k+1$,

$$\begin{aligned} P(x_0) &= \frac{(x_0 - x_{k+1})Q(x_0) - (x_0 - x_0)\widehat{Q}(x_0)}{x_0 - x_{k+1}} = f(x_0), \\ P(x_{k+1}) &= \frac{(x_{k+1} - x_{k+1})Q(x_{k+1}) - (x_{k+1} - x_0)\widehat{Q}(x_{k+1})}{x_0 - x_{k+1}} = f(x_{k+1}). \end{aligned}$$

- **Thus** $P(x)$ interpolates $f(x)$ at $x_0, x_1, \cdots, x_k, x_{k+1}$.

Neville's recursion, $i \leq j$, $(Q_i(c) \stackrel{def}{=} f(x_i))$

$$Q_{i,i+1,\cdots,j,j+1}(x) \stackrel{def}{=} \frac{(x - x_{j+1})Q_{i,i+1,\cdots,j}(x) - (x - x_i)Q_{i+1,\cdots,j,j+1}(x)}{x_i - x_{j+1}}$$

Neville's recursion, $i \leq j$, $(Q_i(c) \stackrel{def}{=} f(x_i))$

$$Q_{i,i+1,\cdots,j,j+1}(x) \stackrel{def}{=} \frac{(x - x_{j+1})Q_{i,i+1,\cdots,j}(x) - (x - x_i)Q_{i+1,\cdots,j,j+1}(x)}{x_i - x_{j+1}}$$

Neville's Method, double-loop

$$
\begin{array}{lllllllll}
f(x_0) & \searrow & & & & & & & \\
 & \longrightarrow & & & & & & & \\
f(x_1) & \searrow & Q_{0,1}(x) & \searrow & & & & & \\
 & \longrightarrow & & \longrightarrow & & & & & \\
f(x_2) & \searrow & Q_{1,2}(x) & \searrow & Q_{0,1,2}(x) & & & & \\
\vdots & & & & & \ddots & & & \\
 & \longrightarrow & & \longrightarrow & & \longrightarrow & & & \\
f(x_{n-2}) & \searrow & Q_{n-3,n-2}(x) & \searrow & Q_{n-4,n-3,n-2}(x) & \searrow & \cdots & & \\
 & \longrightarrow & & \longrightarrow & & \longrightarrow & & & \\
f(x_{n-1}) & \searrow & Q_{n-2,n-1}(x) & \searrow & Q_{n-3,n-2,n-1}(x) & \searrow & \cdots & \searrow & \\
f(x_n) & \longrightarrow & Q_{n-1,n}(x) & \longrightarrow & Q_{n-2,n-1,n}(x) & \longrightarrow & \cdots & \longrightarrow & Q_{0,1,\cdots,n}(x)
\end{array}
$$

Neville's Method, memory re-use version

$$Q_i(x) \overset{def}{\Longleftarrow} \frac{(x - x_{j+1})Q_i(x) - (x - x_i)Q_{i+1}(x)}{x_i - x_{j+1}}$$

Neville's Method, memory re-use version

$$Q_i(x) \overset{def}{\Longleftarrow} \frac{(x - x_{j+1})Q_i(x) - (x - x_i)Q_{i+1}(x)}{x_i - x_{j+1}}$$

Neville's Method, double-loop

$$\begin{array}{lllllllll}
f(x_0) & \searrow & & & & & & & \\
 & \rightarrow & & & & & & & \\
f(x_1) & \searrow & Q_0(x) & \searrow & & & & & \\
 & \rightarrow & & \rightarrow & & & & & \\
f(x_2) & \searrow & Q_1(x) & \searrow & Q_0(x) & & & & \\
\vdots & & & & & \ddots & & & \\
 & \rightarrow & & \rightarrow & & \rightarrow & & & \\
f(x_{n-2}) & \searrow & Q_{n-3}(x) & \searrow & Q_{n-4}(x) & \searrow & \cdots & & \\
 & \rightarrow & & \rightarrow & & \rightarrow & & & \\
f(x_{n-1}) & \searrow & Q_{n-2}(x) & \searrow & Q_{n-3}(x) & \searrow & \cdots & \searrow & \\
f(x_n) & \longrightarrow & Q_{n-1}(x) & \longrightarrow & Q_{n-2}(x) & \longrightarrow & \cdots & \longrightarrow & Q_0(x)
\end{array}$$

Neville's Method, double-loop

- **Step $\underline{1}$**: compute $Q_{i,i+1}(x)$, linear polynomial interpolating $f(x)$ at x_i, x_{i+1}, $i = 0, 1, \cdots, n - \underline{1}$.
- **Step $\underline{2}$**: compute $Q_{i,i+1,i+2}(x)$, quadratic polynomial interpolating $f(x)$ at x_i, x_{i+1}, x_{i+2}, $i = 0, 1, \cdots, n - \underline{2}$.
- ...
- **Step $\underline{j}$**: compute $Q_{i,i+1,\cdots,i+j}(x)$, polynomial of degree j, interpolating $f(x)$ at $x_i, x_{i+1}, \cdots, x_{i+j}$, for $i = 0, 1, \cdots, n - \underline{j}$.
- ...
- **Step $\underline{n}$**: compute $Q_{i,i+1,\cdots,i+n}(x)$, polynomial of degree n, interpolating $f(x)$ at $x_i, x_{i+1}, \cdots, x_{i+n}$, for $i = 0$.

Neville's Method, memory re-use version

- **Step $\underline{\mathbf{0}}$**: $Q_i = f(x_i)$ for $i = 0, 1, \cdots, n$.
- **Step $\underline{1}$**: compute $Q_i \stackrel{def}{=} Q_{i,i+1}(x)$, linear polynomial interpolating $f(x)$ at x_i, x_{i+1}, $i = 0, 1, \cdots, n - \underline{1}$.
- **Step $\underline{2}$**: compute $Q_i \stackrel{def}{=} Q_{i,i+1,i+2}(x)$, quadratic polynomial interpolating $f(x)$ at x_i, x_{i+1}, x_{i+2}, $i = 0, 1, \cdots, n - \underline{2}$.
- $\cdots$
- **Step $\underline{j}$**: compute $Q_i \stackrel{def}{=} Q_{i,i+1,\cdots,i+j}(x)$, polynomial of degree j, interpolating $f(x)$ at $x_i, x_{i+1}, \cdots, x_{i+j}$, for $i = 0, 1, \cdots, n - \underline{j}$.
- $\cdots$
- **Step $\underline{n}$**: compute $Q_i \stackrel{def}{=} Q_{i,i+1,\cdots,i+n}(x)$, polynomial of degree n, interpolating $f(x)$ at $x_i, x_{i+1}, \cdots, x_{i+n}$, for $i = 0$.

Output is Q_0.

Neville's Method, pseudo-code

```
function [y,out] = nev(xx,x,Q)
% Neville's algorithm as a function (save as "nev.m")
%
% inputs:
%    n = order of interpolation (n+1 = # of points)
%    x(1),...,x(n+1)    x coords
%    Q(1),...,Q(n+1)    function values at x
%    xx=evaluation point for interpolating polynomial p
%
%
% On output
%    y     = p(xx):
%    out.Q = intermediate Q values.
%    out.Q(j): approximation by interpolating at points x(j) ... x(n+1)
%
% Written by Ming Gu for Math 128A, Spring 2015
%

N = length(x);
n = N - 1;

for i = n:-1:1
   for j = 1:i
      Q(j) = (xx-x(j))*Q(j+1) - (xx-x(j+N-i))*Q(j);
      Q(j) = Q(j)/(x(j+N-i)-x(j));
   end
end

y = Q(1);
out.Q = Q;
```

§3.3 Divided Differences

- Given $n+1$ distinct points

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

- Interpolating polynomial of degree $\leq n$

$$\begin{aligned} P(x) &= a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \\ &\quad \cdots + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}), \\ \text{with } P(x_0) &= f(x_0), \quad P(x_1) = f(x_1), \quad \cdots, \quad P(x_n) = f(x_n). \end{aligned}$$

It follows

$$\begin{aligned} a_0 &= P(x_0) = f(x_0), \\ \frac{P(x) - f(x_0)}{x - x_0} &= a_1 + a_2(x-x_1) + \\ &\quad \cdots + a_n(x-x_1)\cdots(x-x_{n-1}). \end{aligned}$$

Divided Differences: $f[x_i] = f(x_i), \quad f[x_i, x_{i+1}] = \dfrac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}.$

Let $x = x_1$ in

$$\frac{P(x) - f(x_0)}{x - x_0} = a_1 + a_2(x - x_1) + \cdots + a_n(x - x_1)\cdots(x - x_{n-1}).$$

$$\begin{aligned}
\text{It follows that} \quad a_1 &= \frac{P(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1] \\
\frac{P(x) - P(x_0)}{x - x_0} &= f[x_0, x_1] + a_2(x - x_1) + \cdots \\
&\quad + a_n(x - x_1)\cdots(x - x_{n-1}) \\
\frac{P[x_0, x] - P[x_0, x_1]}{x - x_1} &= a_2 + \cdots + a_n(x - x_2)\cdots(x - x_{n-1}) \\
a_2 &\overset{x=x_2}{=\!=} \frac{P[x_0, x_2] - P[x_0, x_1]}{x_2 - x_1} \overset{def}{=} f[x_1, x_0, x_2]
\end{aligned}$$

Divided Differences: $f[x_i] = f(x_i), \quad f[x_i, x_{i+1}] = \dfrac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}.$

Let $x = x_1$ in

$$\frac{P(x) - f(x_0)}{x - x_0} = a_1 + a_2(x - x_1) + \cdots + a_n(x - x_1)\cdots(x - x_{n-1}).$$

$$\begin{aligned}
\text{It follows that} \quad a_1 &= \frac{P(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1] \\
\frac{P(x) - P(x_0)}{x - x_0} &= f[x_0, x_1] + a_2(x - x_1) + \cdots \\
&\quad + a_n(x - x_1)\cdots(x - x_{n-1}) \\
\frac{P[x_0, x] - P[x_0, x_1]}{x - x_1} &= a_2 + \cdots + a_n(x - x_2)\cdots(x - x_{n-1}) \\
a_2 &\overset{x = x_2}{=\!=} \frac{P[x_0, x_2] - P[x_0, x_1]}{x_2 - x_1} \overset{def}{=} f[x_1, x_0, x_2] \\
&\overset{exercise}{=\!=} f[x_0, x_1, x_2]
\end{aligned}$$

Recursive Divided Differences, for all i, j (common nodes in blue)

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

Recursive Divided Differences, for all i, j (common nodes in blue)

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

then in

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

we have

$$\begin{aligned} a_0 &= f[x_0] \\ a_1 &= f[x_0, x_1] \\ a_2 &= f[x_0, x_1, x_2] \\ \vdots &\quad \vdots \\ a_n &= f[x_0, x_1, \cdots, x_n] \end{aligned}$$

Newton Divided Difference Example

Table 3.10

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

Newton Divided Difference Example

Table 3.10

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

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{0.6200860 - 0.7651977}{1.3 - 1.0} = -0.4837057.$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{-0.5489460 - (-0.4837057)}{1.6 - 1.0} = -0.1087339.$$

Coefficients are numbers on top of all $f[\cdots]$ columns

i	x_i	$f[x_i]$	$f[x_{i-1},x_i]$	$f[x_{i-2},x_{i-1},x_i]$	$f[x_{i-3},\ldots,x_i]$	$f[x_{i-4},\ldots,x_i]$
0	1.0	0.7651977				
			−0.4837057			
1	1.3	0.6200860		−0.1087339		
			−0.5489460		0.0658784	
2	1.6	0.4554022		−0.0494433		0.0018251
			−0.5786120		0.0680685	
3	1.9	0.2818186		0.0118183		
			−0.5715210			
4	2.2	0.1103623				

Newton Divided Difference

Theorem: Suppose that $f \in C^n[a,b]$ and $x_0, x_1, \cdots, x_n$ are distinct nodes in $[a,b]$. Then a number $\xi \in (a,b)$ exists such that

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

Proof: Let $g(x) = f(x) - P(x)$, where

$$\begin{aligned} P(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \\ &\quad \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \end{aligned}$$

Since $g(x_j) = f(x_j) - P(x_j) = 0, \quad j = 0, 1, \cdots, n$, it follows that a number $\xi \in (a,b)$ exists such that $g^{(n)}(\xi) = 0$. But

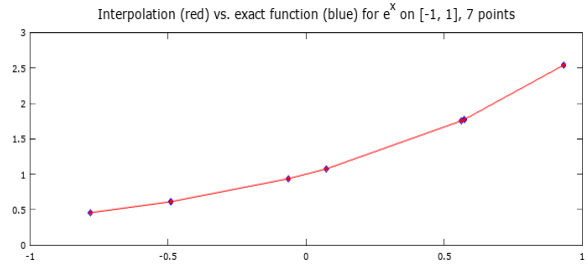
$$\begin{aligned} g^{(n)}(\xi) &= f^{(n)}(\xi) - P^{(n)}(\xi) = f^{(n)}(\xi) - a_n\, n! \\ &= f^{(n)}(\xi) - f[x_0, x_1, \cdots, x_n]\, n! = 0. \end{aligned}$$

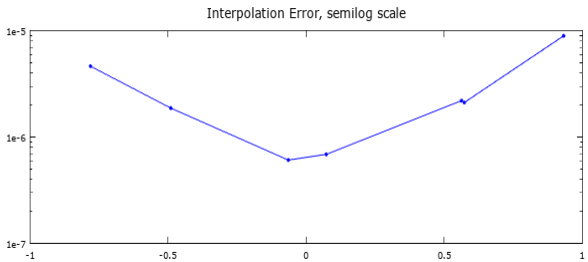
Polynomial Interpolation: Experiments:

- **Easy function**: $f(x) = e^x$ on $[-1, 1]$.

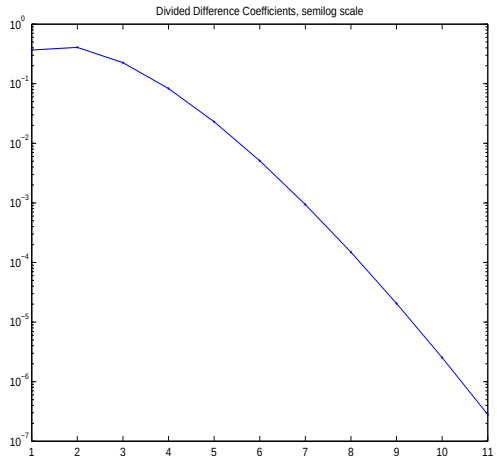
$$|f^{(n)}(\xi)| \leq e \quad \text{for all} \quad \xi \in (-1, 1).$$

7 nodal points ($n = 6$)
random x points.





Divided difference coefficients for e^x on $[-1, 1]$



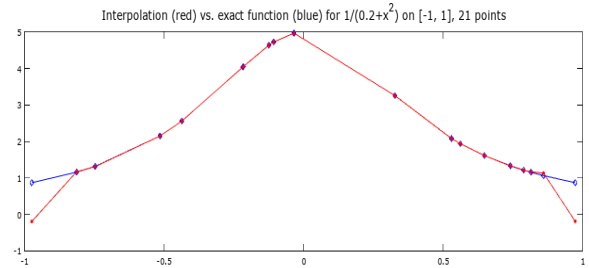
Polynomial Interpolation: Experiments:

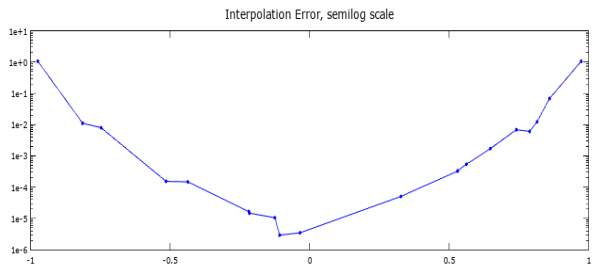
- **Hardest function**: $f(x) = \frac{1}{0.2+x^2}$ on $[-1, 1]$.

$$|f^{(n)}(\xi)| \quad \text{can be very large for some} \quad \xi \in (-1, 1).$$

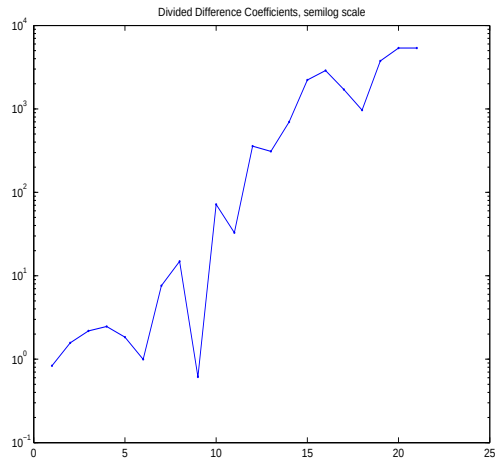
- random x points.

21 nodal points ($n = 20$)
random x points.





Divided difference coefficients for e^x on $[-1, 1]$



Divided difference: data relations

$$
\begin{array}{lllllll}
f[x_0] & \searrow & & & & & \\
 & \rightarrow & & & & & \\
f[x_1] & \searrow & f[x_0,x_1] & \searrow & & & \\
 & \rightarrow & & \rightarrow & & & \\
f[x_2] & \searrow & f[x_1,x_2] & \searrow & f[x_0,x_1,x_2] & & \\
\vdots & & & & & \ddots & \\
 & \rightarrow & & \rightarrow & & \rightarrow & \\
f[x_{n-2}] & \searrow & f[x_{n-3},x_{n-2}] & \searrow & f[x_{n-4},x_{n-3},x_{n-2}] & \searrow & \cdots \\
 & \rightarrow & & \rightarrow & & \rightarrow & \\
f[x_{n-1}] & \searrow & f[x_{n-2},x_{n-1}] & \searrow & f[x_{n-3},x_{n-2},x_{n-1}] & \searrow & \cdots \quad \searrow \\
f[x_n] & \longrightarrow & f[x_{n-1},x_n] & \longrightarrow & f[x_{n-2},x_{n-1},x_n] & \longrightarrow & \cdots \longrightarrow f[x_0,\cdots,x_n]
\end{array}
$$

- computation flow

$$
\begin{array}{lllllll}
f[x_0] & & & & & & \\
f[x_1] & \uparrow & f[x_0, x_1] & & & & \\
f[x_2] & \uparrow & f[x_1, x_2] & \uparrow & f[x_0, x_1, x_2] & & \\
\vdots & & & & & \ddots & \\
f[x_{n-2}] & \uparrow & f[x_{n-3}, x_{n-2}] & \uparrow & f[x_{n-4}, x_{n-3}, x_{n-2}] & \uparrow & \cdots \\
f[x_{n-1}] & \uparrow & f[x_{n-2}, x_{n-1}] & \uparrow & f[x_{n-3}, x_{n-2}, x_{n-1}] & \uparrow & \cdots \\
f[x_n] & \uparrow & f[x_{n-1}, x_n] & \uparrow & f[x_{n-2}, x_{n-1}, x_n] & \uparrow & \cdots \quad f[x_0, \cdots, x_n]
\end{array}
$$

- computation flow

$$
\begin{array}{lllllll}
f[x_0] & & & & & & \\
f[x_1] & \uparrow & f[x_0, x_1] & & & & \\
f[x_2] & \uparrow & f[x_1, x_2] & \uparrow & f[x_0, x_1, x_2] & & \\
\vdots & & & & & \ddots & \\
f[x_{n-2}] & \uparrow & f[x_{n-3}, x_{n-2}] & \uparrow & f[x_{n-4}, x_{n-3}, x_{n-2}] & \uparrow & \cdots \\
f[x_{n-1}] & \uparrow & f[x_{n-2}, x_{n-1}] & \uparrow & f[x_{n-3}, x_{n-2}, x_{n-1}] & \uparrow & \cdots \\
f[x_n] & \uparrow & f[x_{n-1}, x_n] & \uparrow & f[x_{n-2}, x_{n-1}, x_n] & \uparrow & \cdots \quad f[x_0, \cdots, x_n]
\end{array}
$$

- storage (with memory re-use): output is $F_0, F_1, \cdots, F_n$

$$
\begin{array}{lllll}
\boxed{f[x_0] \stackrel{def}{=} F_0} & & & & \\
f[x_1] & \leftarrow & \boxed{f[x_0, x_1] \stackrel{def}{=} F_1} & & \\
f[x_2] & \leftarrow & f[x_1, x_2] & \leftarrow & \boxed{f[x_0, x_1, x_2] \stackrel{def}{=} F_2} \\
\vdots & & & & \ddots
\end{array}
$$

Divided difference, $O(n)$ memory

```
function F = NDD1(x,f)
%
% This function implements Newton's Divided Difference Algorithm
% F is the vector of coefficients
%
% Updated by Ming Gu for Math 128A, Spring 2015
%
N = length(x);
F = f;
for k=2:N
    for j = N:-1:k
       F(j) = (F(j)-F(j-1))/(x(j)-x(j-k+1));
    end
end
```

Divided difference, $O(n)$ memory

```
function F = NDD1(x,f)
%
% This function implements Newton's Divided Difference Algorithm
% F is the vector of coefficients
%
% Updated by Ming Gu for Math 128A, Spring 2015
%
N = length(x);
F = f;
for k=2:N
    for j = N:-1:k
       F(j) = (F(j)-F(j-1))/(x(j)-x(j-k+1));
    end
end
```

book version, $O(n^2)$ memory

```
function F = NewtonDividedDifference(x,f)
%
% This function implements Newton's Divided Difference Algorithm
% F is the vector of coefficients
%
% Written by Ming Gu for Math 128A, Fall 2008
%
n = length(x);
P = diag(f);
for k=2:n
   for j = k-1:-1:1
       P(k,j) = (P(k,j+1)-P(k-1,j))/(x(k)-x(j));
   end
end
F = P(:,1);
```