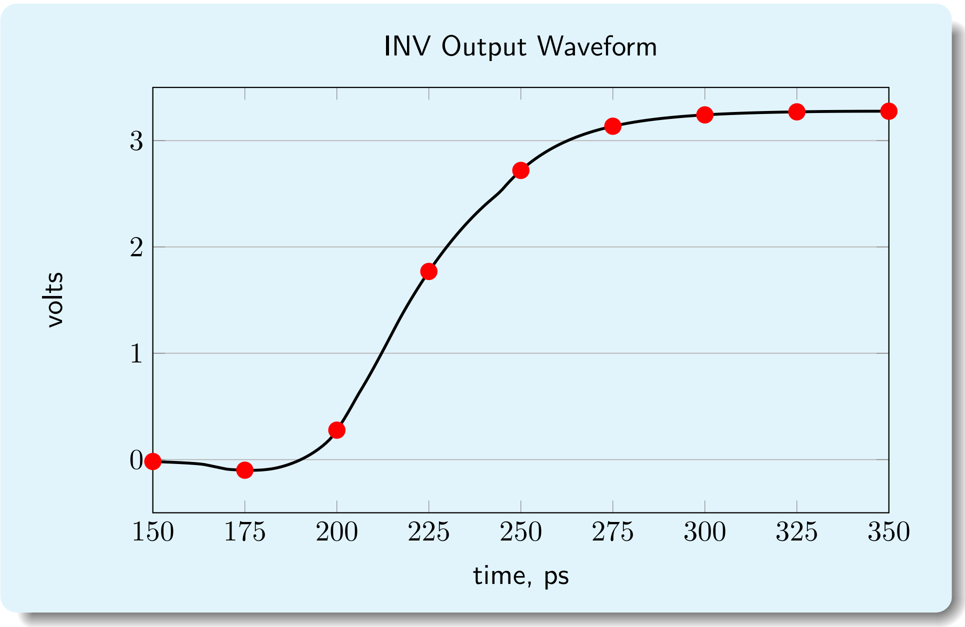
# Unit 5 Interpolation

Numerical Analysis

Apr. 16, 2015



## Data and Functions




- In real world, one uses limited data points to represent a real math function
  - Can one get the function values in-between data points accurately?
  - Or to find the underlying function given the limited data points.
  - Interpolation problems

## Interpolation Problems

### Definition 5.1.1. Interpolation problem

Given a set of $n+1$ **support points**

$$\{(x_i, y_i)\}, i = 0, 1, \cdots, n, \text{ with } x_j \neq x_k \text{ for } j \neq k, \tag{5.1.1}$$

find the function $F(x\,; a_0, \cdots, a_n)$ with $n+1$ coefficients, $a_0, a_1, \cdots, a_n$, such that

$$F(x_i\,; a_0, \cdots, a_n) = y_i, \qquad i = 0, \cdots, n. \tag{5.1.2}$$

### Definition 5.1.2.

Given the interpolation problem as in the definition above we have the followings:

**Support abscissas**: $\{x_i\}$,

**Support ordinates**: $\{y_i\}$.

**Linear interpolation**: if $F$ can be expressed as

$$F(x\,; a_0, \cdots, a_n) = a_0 F_0(x) + a_1 F_1(x) + \cdots + a_n F_n(x).$$

**Trigonometric interpolation**: if $F$ can be expressed as

$$F(x\,; a_0, \cdots, a_n) = a_0 F_0(x) + a_1 e^{xi} + a_2 e^{2xi} + \cdots + a_n e^{nxi}, \quad \text{with } i^2 = -1.$$



## Interpolation of Polynomials

### Definition 5.1.3.

The symbol $\Pi_n$ denotes the set of all polynomials of order not greater than $n$.

### Definition 5.1.4. Polynomial interpolation

Given the n+1 support points, find $F(x\,; a_0, \cdots, a_n) \in \Pi_n$

$$F(x\,; a_0, \cdots, a_n) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

such that Eq. (5.1.2) is satisfied, then it is a polynomial interpolation problem.

- Note that there are $n+1$ support points, the order of $F$ cannot be greater than $n$.

# Interpolation of Polynomials – Example

## Example 5.1.5.

Find $F(x) \in \Pi_2$ such that $F(0) = 2,\ F(1) = 1,\ F(2) = 2.$

- Answer: $F(x) = x^2 - 2x + 2.$
- Note that $F(x) = a_0 + a_1 x + a_2 x^2$ can be found with the constraints

$$\begin{aligned} F(0) &= a_0 + a_1 \cdot 0 + a_2 \cdot 0^2 = 2 \\ F(1) &= a_0 + a_1 \cdot 1 + a_2 \cdot 1^2 = 1 \\ F(2) &= a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 = 2 \end{aligned}$$

Or

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}$$

Solution: $a_0 = 2,\ a_1 = -2,\ a_2 = 1.$

- Given $n+1$ support points $\{(x_i, y_i), 0 \le i \le n\}$, the system of equations can be formulated easily and the solution found.
- Note that with the condition, $x_j \ne x_k$ if $j \ne k$, then the system is non-singular and there is only one solution.



# Interpolation of Polynomials – Lagrange Interpolation

- The solution can also be found using Lagrange Interpolation Formula

$$\begin{aligned} F(x) &= F(0)\frac{(x-1)(x-2)}{(0-1)(0-2)} + F(1)\frac{(x-2)(x-0)}{(1-2)(1-0)} + F(2)\frac{(x-0)(x-1)}{(2-0)(2-1)} \\ &= 2\frac{(x-1)(x-2)}{2} - x(x-2) + 2\frac{x(x-1)}{2} \\ &= (x-1)(x-2) - x(x-2) + x(x-1) = x^2 - 2x + 2 \end{aligned}$$

## Definition 5.1.6. Lagrange Interpolation Formula

Given support points $\{(x_i, y_i), 0 \le i \le n\}$, then the Lagrange interpolation formula is

$$F(x) = \sum_{i=0}^{n} y_i \prod_{k=0, k\ne i}^{n} \frac{x - x_k}{x_i - x_k}. \tag{5.1.3}$$

Or let

$$L_i(x) = \prod_{k=0, k\ne i}^{n} \frac{x - x_k}{x_i - x_k} \tag{5.1.4}$$

then

$$F(x) = \sum_{i=0}^{n} y_i L_i(x). \tag{5.1.5}$$

# Lagrange Interpolation Formula

- Note that

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

And we have

$$L_i(x_i) = 1, \quad (5.1.6)$$
$$L_i(x_j) = 0, \text{ if } i \neq j. \quad (5.1.7)$$

Thus $F(x_i) = y_i$ always holds. Since the degrees of Eqs. (5.1.3), (5.1.4), (5.1.5) are all $n$, the Lagrange Interpolation Formula is the solution to the polynomial interpolation problem.

**Theorem 5.1.7.**

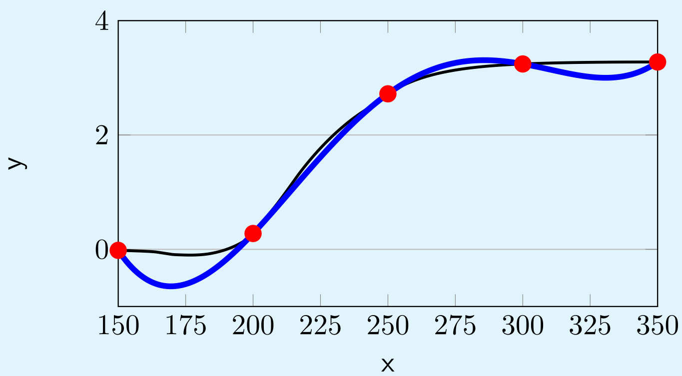
Given $n+1$ support points, $\{(x_i, y_i), 0 \leq i \leq n\}$ with $x_i \neq x_j$ if $i \neq j$, then there exists a unique polynomial $F \in \Pi_n$ with

$$F(x_i) = y_i, 0 \leq i \leq n.$$

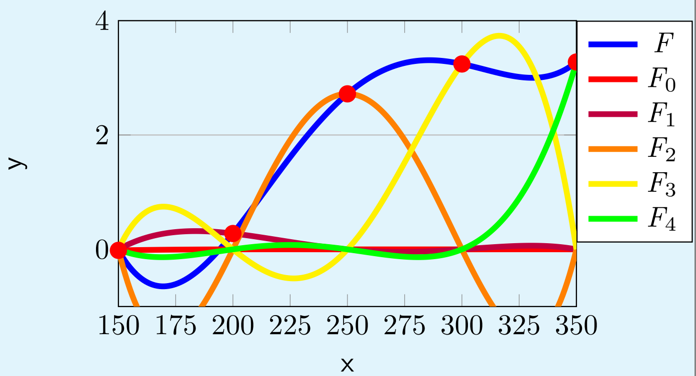


# Interpolation of Polynomials, $n = 4$ Case

**Lagrange Interpolation**

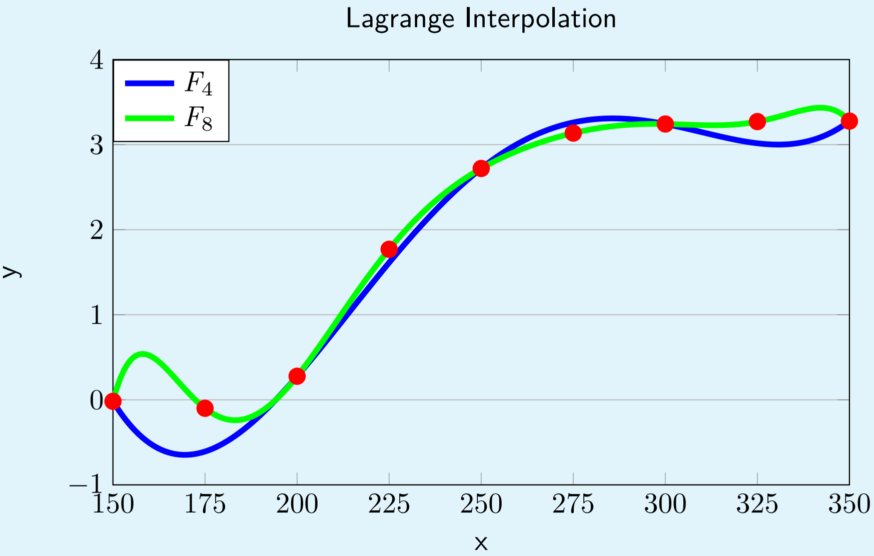



**Lagrange, $L_i$**




- In the right figure, $F_i = y_i L_i(x)$.
- Note that
  - $F(x_i) = y_i$, $0 \leq i \leq 4$.
  - Between support points, the function can be different than one's expectation.
    - Especially for small $x$ and large $x$.

# Interpolation of Polynomials

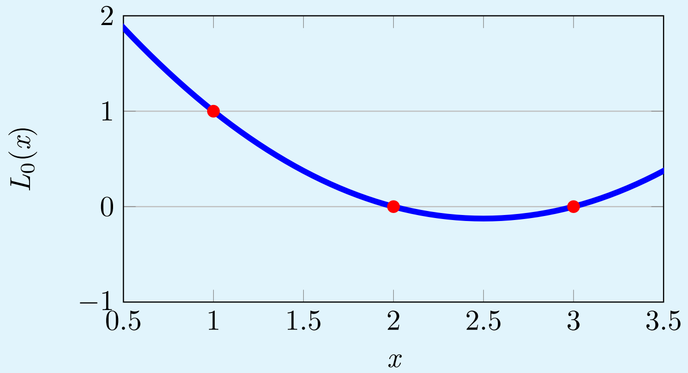



- Note that
  - Higher order interpolations (more support points) the interpolation is more accurate.
  - But it is relatively less accurate for the regions closer to $x_0$ and $x_n$.
  - It is not a good idea to use this formula for extrapolation.

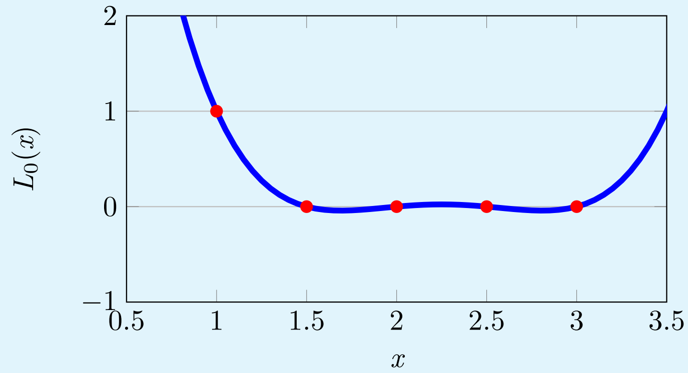


# Lagrange Formula Plots

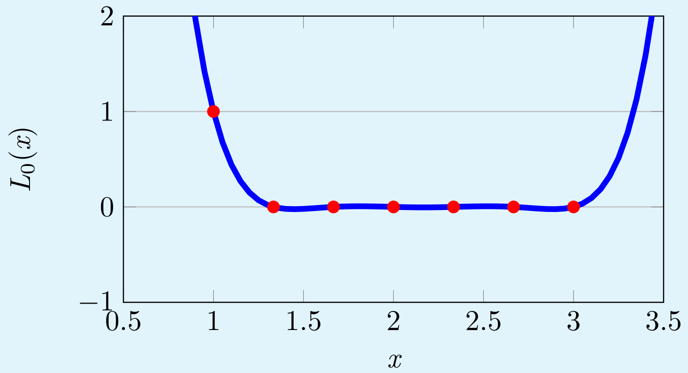
## n=2, $L_0$



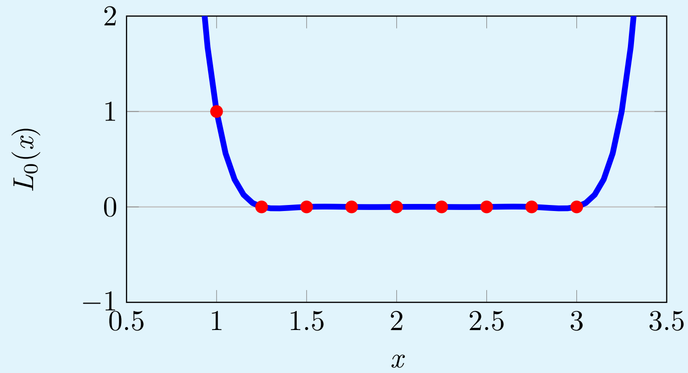

## n=4, $L_0$




## n=6, $L_0$

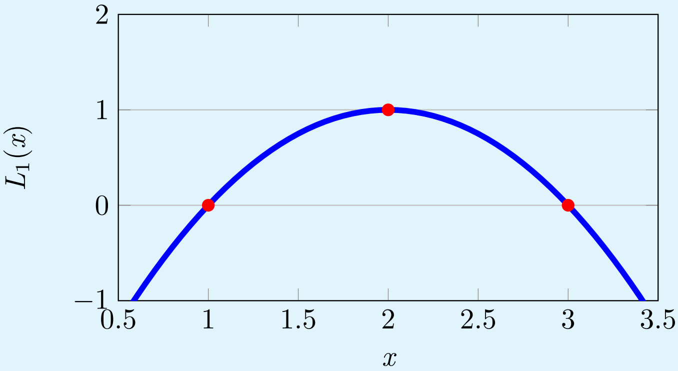



## n=8, $L_0$

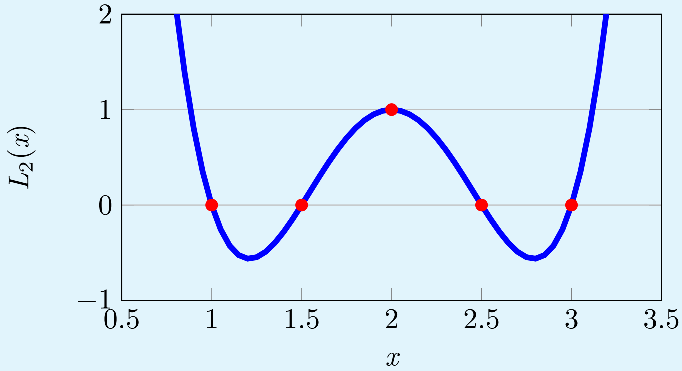


# Lagrange Formula Plots, II

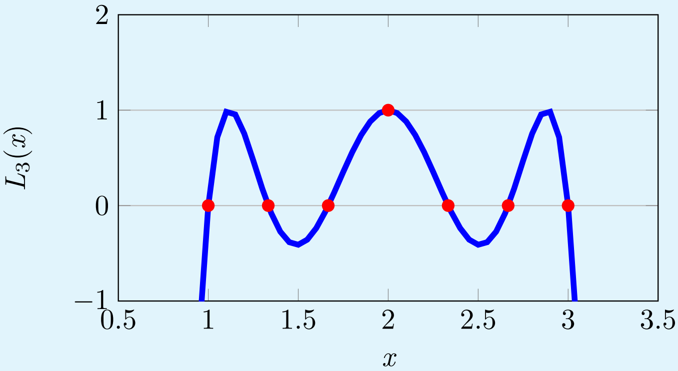
## n=2, $L_1$



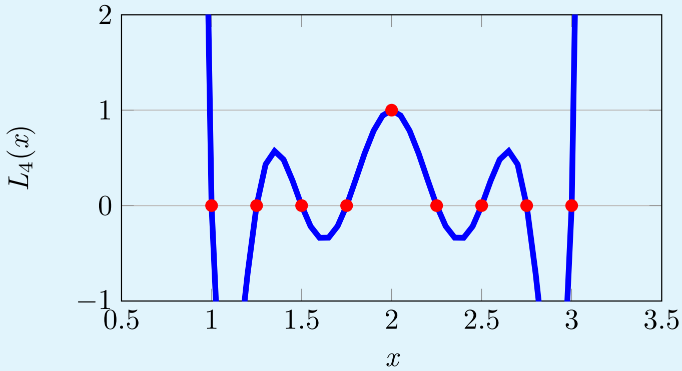

## n=4, $L_2$




## n=6, $L_3$



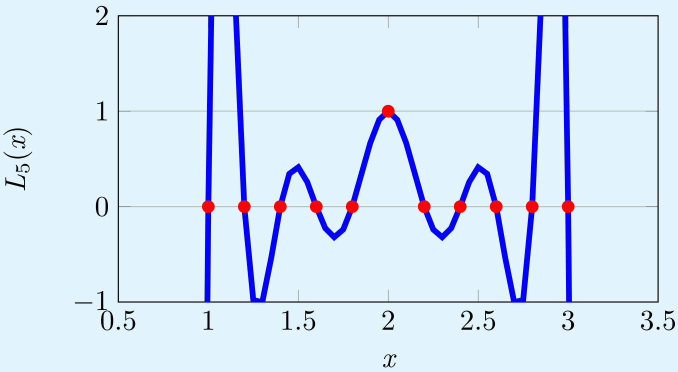

## n=8, $L_4$



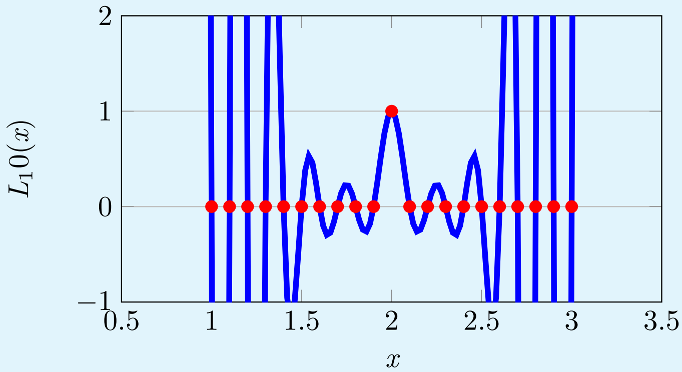



# Lagrange Formula Plots, III

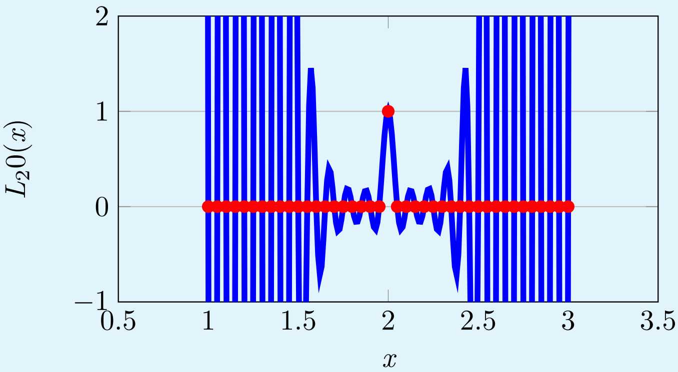
## n=10, $L_5$




## n=20, $L_{10}$



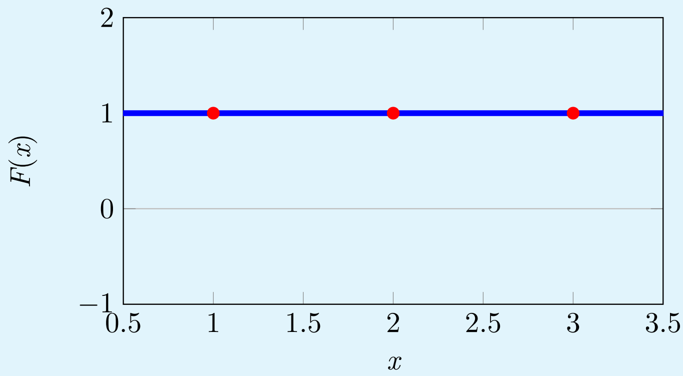

## n=40, $L_{20}$



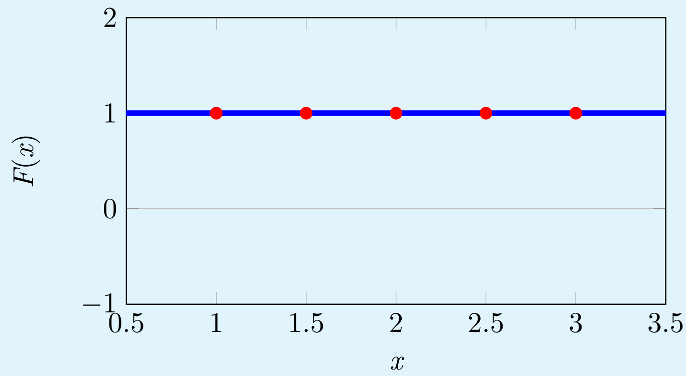

- $L_i(x_j) = \delta_{i,j}$ for $x_i$, $0 \leq i \leq n$
- $L_i(x)$, $x \neq x_i$, is relatively small in the vicinity of $x_i$
  - But it can be large for small $x$ and large $x$

# Lagrange Formula Plots, $y_i = 1$

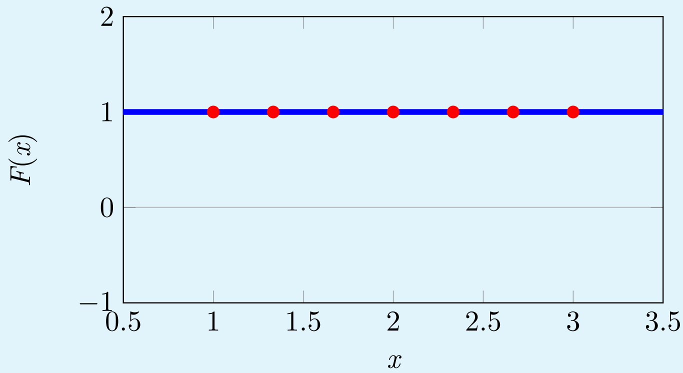
### n=2, $F$



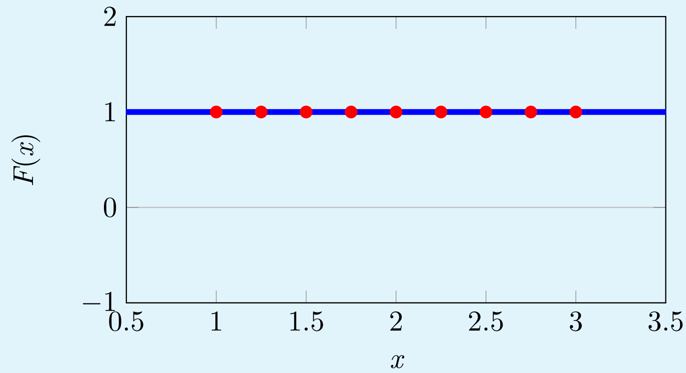

### n=4, $F$




### n=6, $F$



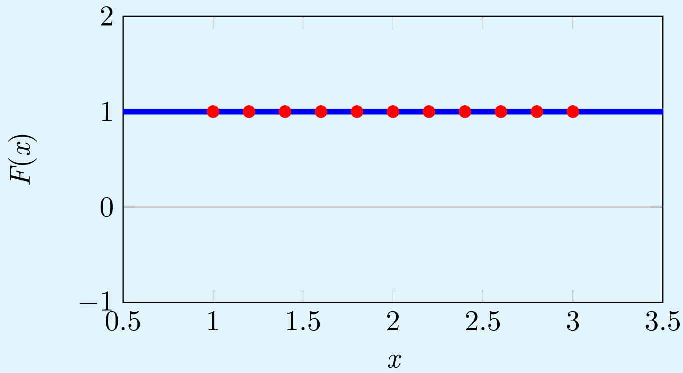

### n=8, $F$



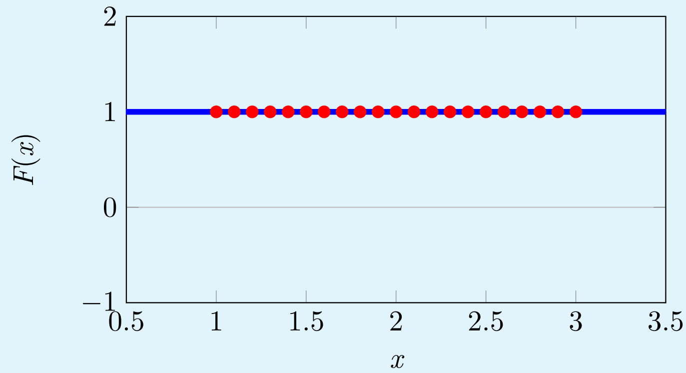



# Lagrange Formula Plots, $y_i = 1$, II

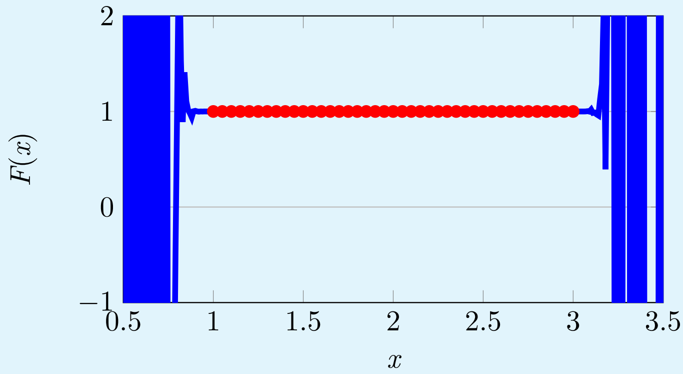
### n=10, $F$




### n=20, $F$




### n=40, $F$




- Sum of $L_i$ reproduces $y_j = F(x_j)$ when $F$'s order is low
  - If the data can be represented using polynomial of order less than $n$, then the Lagrange Interpolation should give exact solution
- For large $n$, watch out for numerical errors

# Simplifying Calculation - Example

- In the following, we use the notation

$$F_{i_0 i_1 \cdots i_k}(x) = \sum_{k=i_0, i_1, \cdots, i_k} y_k L_k(x)$$

- Example with 3 support points, $\{(x_0, y_0), (x_1, y_1), (x_2, y_2)\}$, the Lagrange interpolation formula is

$$F_{012}(x) = y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_2)(x-x_0)}{(x_1-x_2)(x_1-x_0)} + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

- And

$$\begin{aligned}
&\frac{(x-x_0)F_{12}(x) - (x-x_2)F_{01}(x)}{x_2-x_0} = \frac{x-x_0}{x_2-x_0} F_{12}(x) - \frac{x-x_2}{x_2-x_0} F_{01}(x) \\
&= \frac{x-x_0}{x_2-x_0}\left(y_1 \frac{x-x_2}{x_1-x_2} + y_2 \frac{x-x_1}{x_2-x_1}\right) - \frac{x-x_2}{x_2-x_0}\left(y_0 \frac{x-x_1}{x_0-x_1} + y_1 \frac{x-x_0}{x_1-x_0}\right) \\
&= y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} + y_1 \frac{(x-x_0)(x-x_2)}{x_2-x_0}\left(\frac{1}{x_1-x_2} - \frac{1}{x_1-x_0}\right) \\
&\quad + y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} \\
&= F_{012}(x)
\end{aligned}$$



# Neville's Algorithm

- Thus

$$F_{012}(x) = \frac{(x-x_0)F_{12}(x) - (x-x_2)F_{01}(x)}{x_2-x_0}$$

- In general, it can be shown

$$F_{i_0 i_1 \cdots i_k}(x) = \frac{(x-x_{i_0})F_{i_1 i_2 \cdots i_k}(x) - (x-x_{i_k})F_{i_0 i_1 \cdots i_{k-1}}(x)}{x_{i_k}-x_{i_0}}. \tag{5.1.8}$$

## Theorem 5.1.8. Neville's Algorithm

Given $n+1$ support points $\{(x_i, y_i)\}$, $i = 0, \cdots, n$, with $x_j \neq x_k$ if $j \neq k$, then the Lagrange interpolation at the point $x$, $F_{01\cdots n}(x)$, can be calculated using the following recursion formula:

$$F_i(x) = y_i, \tag{5.1.9}$$

$$F_{i_0 i_1 \cdots i_k}(x) = \frac{(x-x_{i_0})F_{i_1 i_2 \cdots i_k}(x) - (x-x_{i_k})F_{i_0 i_1 \cdots i_{k-1}}(x)}{x_{i_k}-x_{i_0}}. \tag{5.1.10}$$

# Neville's Algorithm – Implementation

- Neville's algorithm is a recursion formula and can be implemented using recursive function directly
- Assuming the support points are stored in two arrays XS and YS then following function calculates Lagrange interpolation at point $x$ using Neville's algorithm

**Algorithm 5.1.9. Neville's Algorithm**

```
double NEV(double x,double XS[],double YS[],int i0,int ik)
{
    if (i0==ik) return YS[i0];
    else return
        ((x-XS[i0])*NEV(x,XS,YS,i0+1,ik)
         -(x-XS[ik])*NEV(x,XS,YS,i0,ik-1))/(XS[ik]-XS[i0]);
}
```

- For example (5.1.5), `XS[3]={0,1,2}`, `YS[3]={2,1,2}` and `NEV(x,XS,YS,0,2)` calculates the value of Lagrange Interpolation formula at $x$.



# Neville's Algorithm Evaluation

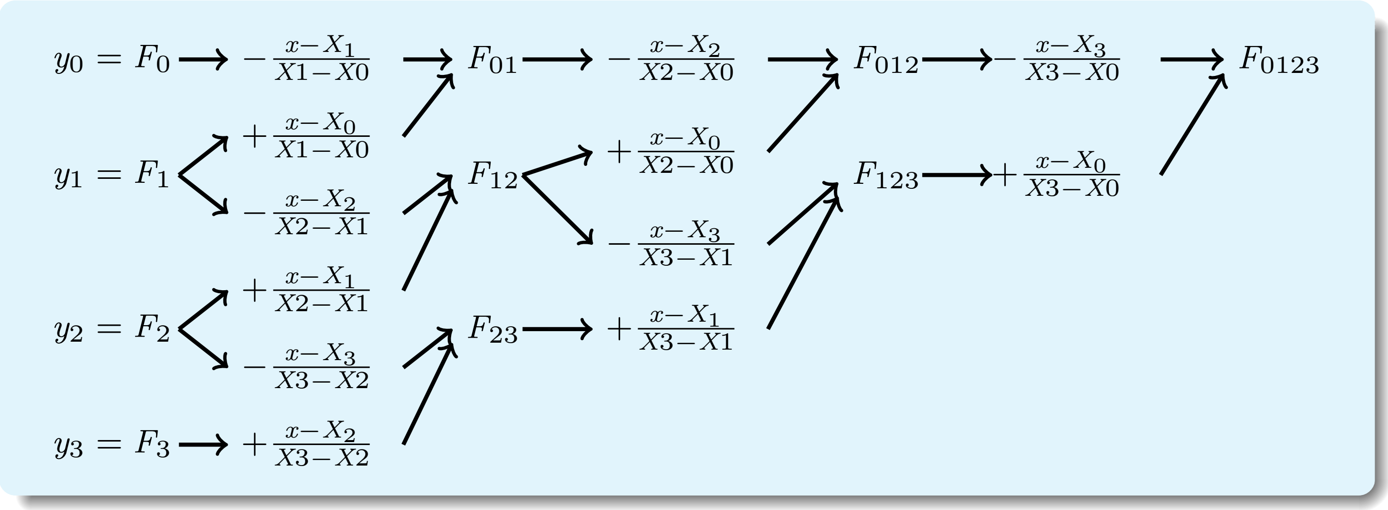
- The recursive form of Neville's algorithm is not the most efficient implementation.
- For example, with 4 support points, Neville's algorithm expands to

$F_{0123}$
- $-\frac{x-X_3}{X_3-X_0}F_{012}$
  - $-\frac{x-X_2}{X_2-X_0}F_{01}$
    - $-\frac{x-X_1}{X_1-X_0}F_0$
    - $+\frac{x-X_0}{X_1-X_0}F_1$
  - $+\frac{x-X_0}{X_2-X_0}F_{12}$
    - $-\frac{x-X_2}{X_2-X_1}F_1$
    - $+\frac{x-X_1}{X_2-X_1}F_2$
- $+\frac{x-X_0}{X_3-X_0}F_{123}$
  - $-\frac{x-X_3}{X_3-X_1}F_{12}$
    - $-\frac{x-X_2}{X_2-X_1}F_1$
    - $+\frac{x-X_1}{X_2-X_1}F_2$
  - $+\frac{x-X_1}{X_3-X_1}F_{23}$
    - $-\frac{x-X_3}{X_3-X_2}F_2$
    - $+\frac{x-X_2}{X_3-X_2}F_3$

- Many repeated evaluations were performed
- Total number of function calls is $2^{n+1}-1$ for $n+1$ support points

# Improving Neville's Algorithm

- Note that Neville's evaluation sequence can be rearranged as the following




- In this way, the number of evaluation is reduced to $\frac{(n+1)(n+2)}{2}$
  - Very efficient, around 2X faster than Lagrange interpolation formula
- Furthermore, all values of $F_{i\cdots k}$ can be stored in the same array NS



# Non-recursive Neville's Algorithm

- Assuming NS stores the temporary values of $F(x)$, Neville's algorithm can be rewritten in the following non-recursive form

**Algorithm 5.1.10. Non-recursive Neville's Algorithm**

```
double NEV(double x,double XS[],double YS[],int n)
{
    double NS[n];
    int i,j,k;
    for (i=0; i<n; i++) NS[i]=YS[i];
    for (k=1; k<n; k++) {
        for (j=0; j<n-k; j++) {
            NS[j]=((x-XS[j])*NS[j+1]-(x-XS[k+j])*NS[j])
                  /(XS[j+k]-XS[j]);
        }
    }
    return NS[0];
}
```

- The argument $n$ is the number of support points
  - Instead of $n+1$ support points

# Neville's Algorithm - the 2nd Form

- The equation for Neville's algorithm, Eq. (5.1.10), can be rewritten as

$$
\begin{aligned}
F_{i_0 i_1 \cdots i_k}(x) &= F_{i_1 i_2 \cdots i_k}(x) + \frac{F_{i_1 i_2 \cdots i_k}(x) - F_{i_0 i_1 \cdots i_{k-1}}(x)}{\frac{x - x_{i_0}}{x - x_{i_k}} - 1} \qquad (5.1.11)\\
&= F_{i_1 i_2 \cdots i_k}(x) + \frac{(F_{i_1 i_2 \cdots i_k}(x) - F_{i_0 i_1 \cdots i_{k-1}}(x))(x - x_{i_k})}{x_{i_k} - x_{i_0}}\\
&= \frac{F_{i_1 i_2 \cdots i_k}(x)(x_{i_k} - x_{i_0} + x - x_{i_k}) - F_{i_0 i_1 \cdots i_{k-1}}(x)(x - x_{i_k})}{x_{i_k} - x_{i_0}}\\
&= \frac{(x - x_{i_0})F_{i_1 i_2 \cdots i_k}(x) - (x - x_{i_k})F_{i_0 i_1 \cdots i_{k-1}}(x)}{x_{i_k} - x_{i_0}}
\end{aligned}
$$

- This leads to a slightly different implementation
  - Recursive version is straightforward
  - Non-recursive version is also straightfoward



# Neville's Algorithm - the 3rd Form

- The 3rd form of Neville's algorithm can be defined as following
- Given $n+1$ support points $\{(x_i, y_i), 0 \le i \le n\}$, let

$$
\begin{aligned}
Q_i(x) &= D_i(x) = y_i \qquad (5.1.12)\\
Q_{i_0 i_1 \cdots i_k}(x) &= F_{i_0 i_1 \cdots i_k}(x) - F_{i_1 i_2 \cdots i_k}(x)\\
D_{i_0 i_1 \cdots i_k}(x) &= F_{i_0 i_1 \cdots i_k}(x) - F_{i_0 i_1 \cdots i_{k-1}}(x)
\end{aligned}
$$

  - Note that $Q_{i_0 i_1 \cdots i_k}(x)$ is the difference of two polynomial interpolations; one for the support points $\{(x_i, y_i), 0 \le i \le k\}$ and the other for the support points $\{(x_i, y_i), 1 \le i \le k\}$. The order of the first polynomial is $k$; while the latter is $k-1$.
  - $D_{i_0 i_1 \cdots i_k}(x)$ is also the difference of polynomials of two sets of support points. And their orders differ by 1 also.
- Then

$$
\begin{aligned}
&Q_{i_0 i_1 \cdots i_n}(x) + Q_{i_1 i_2 \cdots i_n}(x) + \cdots + Q_{i_{n-1} i_n}(x) + Q_{i_n}(x)\\
=&F_{i_0 i_1 \cdots i_n}(x) - F_{i_1 i_2 \cdots i_n}(x) + F_{i_1 i_2 \cdots i_n}(x) - F_{i_2 i_3 \cdots i_n}(x) + \cdots\\
&+ F_{i_{n-1} i_n}(x) - F_{i_n}(x) + F_{i_n}(x)\\
=&F_{i_0 i_1 \cdots i_n}(x)
\end{aligned}
$$

# Neville's Algorithm - the 3rd Form, II

- Thus,

$$\sum_{j=0}^{n} Q_{i_j \cdots i_n}(x) = F_{i_0 i_1 \cdots i_n}(x). \tag{5.1.13}$$

- Furthermore, we also have

$$Q_{i_0 \cdots i_k}(x) = \left[D_{i_1 \cdots i_k}(x) - Q_{i_0 \cdots i_{k-1}}(x)\right] \frac{x_i - x}{x_{i-k} - x_i} \tag{5.1.14}$$

$$D_{i_0 \cdots i_k}(x) = \left[D_{i_1 \cdots i_k}(x) - Q_{i_0 \cdots i_{k-1}}(x)\right] \frac{x_{i-k} - x}{x_{i-k} - x_i} \tag{5.1.15}$$

- Combining Eqs (5.1.12, 5.1.13, 5.1.14, 5.1.15) we have the 3rd form of Neville's algorithm
- This form improves the accuracy of the interpolation since the difference of polynomials are calculated and then summed up



# Newton's Interpolation Formula

- Neville's algorithm can calculate a single interpolated value $F(x)$ rather than the interpolating formula.
- Newton's interpolation formula can calculate the interpolating polynomial
- Given the $n+1$ support points $\{(x_i, y_i)\}$, $0 \le i \le n$ with $x_j \ne x_k$ if $j \ne k$, the interpolating polynomial is assumed to have the following form

$$\begin{aligned} F(x) &= F_{01\cdots n}(x) \\ &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots \\ &\quad + a_n(x - x_0)\cdots(x - x_{n-1}). \end{aligned} \tag{5.1.16}$$

- Thus, we have

$$\begin{aligned} y_0 &= F(x_0) = a_0 \\ y_1 &= F(x_1) = a_0 + a_1(x_1 - x_0) \\ y_2 &= F(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ &\cdots \end{aligned}$$

# Newton's Interpolation Formula, II

- The coefficients can be calculated as following

$$\begin{aligned}
a_0 &= y_0 \\
a_1 &= \frac{y_1 - a_0}{x_1 - x_0} \\
a_2 &= \frac{y_2 - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\
&\cdots \\
a_n &= \frac{y_n - a_0 - \cdots - a_{n-1}\prod_{i=0}^{n-2}(x_n - x_i)}{\prod_{i=0}^{n-1}(x_n - x_i)}
\end{aligned} \tag{5.1.17}$$

- It needs $n(n-1)$ multiplications and $n-1$ divisions to calculate all coefficients.



# Divided Difference

- Let $F_{i_0 i_1 \cdots i_{k-1}}(x)$ be the polynomial of the support points $\{(x_{i_j}, y_{i_j})\}$, $j = 0, \cdots, k-1$, and $F_{i_0 i_1 \cdots i_k}(x)$ be the polynomial of the support points $\{(x_{i_j}, y_{i_j})\}$, $j = 0, \cdots, k$. Then there is a unique coefficient $a_{i_0 i_1 \cdots i_k}$ such that

$$F_{i_0 i_1 \cdots i_k}(x) = F_{i_0 i_1 \cdots i_{k-1}}(x) + a_{i_0 i_1 \cdots i_k}(x - x_{i_0})(x - x_{i_1}) \cdots (x - x_{i_{k-1}}).$$

- And thus

$$\begin{aligned}
F_{i_0 i_1 \cdots i_k}(x) = a_{i_0} &+ a_{i_0 i_1}(x - x_{i_0}) + \cdots \\
&+ a_{i_0 i_1 \cdots i_k}(x - x_{i_0})(x - x_{i_1}) \cdots (x - x_{i_{k-1}}).
\end{aligned}$$

- Note that

$$\begin{aligned}
F_{i_0 i_1 \cdots i_{k-1}}(x) &= a_{i_0} + a_{i_0 i_1}(x - x_{i_0}) + \cdots \\
&\quad + a_{i_0 i_1 \cdots i_{k-1}}(x - x_{i_0})(x - x_{i_1}) \cdots (x - x_{i_{k-2}}) \\
F_{i_1 i_2 \cdots i_k}(x) &= a_{i_1} + a_{i_1 i_2}(x - x_{i_1}) + \cdots \\
&\quad + a_{i_1 i_2 \cdots i_{k-1}}(x - x_{i_1})(x - x_{i_2}) \cdots (x - x_{i_{k-1}})
\end{aligned}$$

  - Both are polynomial interpolation formulas and (5.1.10) applies

## Divided Difference, II

- And
$$(x_{i_k} - x_{i_0}) F_{i_0 i_1 \cdots i_k}(x) = (x - x_{i_0}) F_{i_1 i_2 \cdots i_k}(x) - (x - x_{i_k}) F_{i_0 i_1 \cdots i_{k-1}}(x)$$
- Compare the coefficient of the $x^k$ term
$$(x_{i_k} - x_{i_0}) a_{i_0 i_1 \cdots i_k} = a_{i_1 i_2 \cdots i_k} - a_{i_0 i_1 \cdots i_{k-1}}$$
- Thus
$$a_{i_0 i_1 \cdots i_k} = \frac{a_{i_1 i_2 \cdots i_k} - a_{i_0 i_1 \cdots i_{k-1}}}{x_{i_k} - x_{i_o}} \tag{5.1.18}$$
  - This is the $k$*'th divided differences*.
- Since this divided difference is uniquely determined by the $k$ support points, it is invariant to the permutation of the support points.

**Theorem 5.1.11.**

The divided differences $a_{i_0 i_1 \cdots i_k}$ are invariant to permutations of the indices $i_0, i_1, \cdots, i_k$. That is, if
$$(j_0, j_1, \cdots, j_k) = (i_{s_0}, i_{s_1}, \cdots, i_{s_k})$$
is a permutation of the indices $i_0, i_1, \cdots, i_k$m then
$$a_{j_0, j_1, \cdots, j_k} = a_{i_0, i_1, \cdots, i_k}.$$



## Divided Differences – Implementation

- The coefficients of Eq. (5.1.16) can be calculated efficiently using the following algorithm.

**Algorithm 5.1.12. Divided Difference**

```
double DDif(double XS[],double YS[],double A[],int i0,int ik)
{
  double result;
  if (i0==ik) result=YS[i0];
  else {
    result=(DDif(XS,YS,A,i0+1,ik)-DDif(XS,YS,A,i0,ik-1))
      /(XS[ik]-XS[i0]);
  }
  if (i0==0) A[ik]=result;
  return result;
}
```

- After executing `DDif(XS,YS,A,0,n)`, the array element `A[k]` contains the *k'th divided difference*.
- This algorithm is more efficient than the direction implementation of Eq. (5.1.17) – Twice faster.

# Divided Differences Function

- The divided differences is a useful function in numerical analysis and it is defined as following.

**Definition 5.1.13. Divided differences.**

Given a function $f\colon \mathbb{R} \to \mathbb{R}$ and a set $\{x_i\}$, $x_i \in \mathbb{R}$, the divided differences is

$$\begin{aligned} f[x_i] &= f(x_i), \\ f[x_0, x_1, \cdots, x_k] &= \frac{f[x_1, x_2, \cdots, x_k] - f[x_0, x_1, \cdots, x_{k-1}]}{x_k - x_0} \end{aligned} \tag{5.1.19}$$

- Example:

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



# Divided Differences Function – Properties

**Theorem 5.1.14.**

The divided difference $f[x_0, x_1, \cdots, x_k]$ is invariant to the permutation of $x_0, x_1, \cdots, x_k$.

**Theorem 5.1.15.**

If $f(x)$ is a polynomial of degree $N$, then

$$f[x_0, x_1, \cdots, x_k] = 0$$

for $k > N$.

- With the definitions of divided differences, the Newton Interpolation formula can be written as

$$F_{i_0 i_1 \cdots i_n}(x) = f[x_{i_0}] + f[x_{i_0}, x_{i_1}](x - x_{i_0}) + \cdots + f[x_{i_0}, x_{i_1}, \cdots x_{i_n}](x - x_{i_0}) \cdots (x - x_{i_{n-1}}) \tag{5.1.20}$$

# Divided Differences and Derivatives

- By definition,
$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$
If $x_1 = x_0 + h$ and $h \ll 1$
$$f[x_0, x_0 + h] = \frac{f(x_0 + h) - f(x_0)}{h} \longrightarrow f'(x_0) \qquad \text{as } h \to 0$$
Thus, $f[x_0, x_0] = f'(x_0)$
- Next, let $x_1 = x_0 + h$ and $x_2 = x_1 + h = x_0 + 2h$
$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= \frac{f[x_0 + h, x_0 + 2h] - f[x_0, x_0 + h]}{2h} \\ &\sim \frac{f'(x_0 + h) - f'(x_0)}{2h} \end{aligned}$$
Thus, $f[x_0, x_0, x_0] = \dfrac{f''(x_0)}{2}$
- It can be shown that
$$f[x_0, x_1, \cdots, x_k] \sim \frac{f^{(k)}(x_0)}{k!} \qquad \text{if } x_0 = x_1 = \cdots = x_k$$



# Newton's Interpolation Formula

- Newton's interpolation formula can be written as
$$\begin{aligned} F(x) &= F_{01\cdots n}(x) \\ &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ &\qquad + f[x_0, x_1, \cdots, x_n](x - x_0)\cdots(x - x_{n-1}) \end{aligned} \tag{5.1.21}$$
Compare that to Taylor Series
$$F(x) = x_0 + f'(x_0)(x - x_0) + \frac{f''}{2}(x - x_0)^2 + \cdots + \frac{f^{(n)}}{n!}(x - x_0)^n$$
When the support abscissas has $x$'s close to each other then the Newton inpolation formula approaches Taylor series expansion.

# Error in Polynomial Interpolation

- To study the error in polynomial approximation, we assume the underlying function, $f$, of the support points, $\{(x_i, y_i)\}$, $0 \le i \le n$, is known and the error is defined as
$$f(x) - F_{01\cdots n}(x) \tag{5.1.22}$$
- Note that $f(x_i) = y_i$.
- And when $x = x_i$, $0 \le i \le n$, the error is zero since $F_{01\cdots n}$ is a polynomial interpolation of the support points.
- Suppose one wants to find the error at $x = \bar{x}$, i.e., $f(\bar{x}) - F_{01\cdots n}(\bar{x})$, let's define

$$x_m = \min\{x\}, x \in \{x_0, x_1, \cdots x_n, \bar{x}\},$$
$$x_M = \max\{x\}, x \in \{x_0, x_1, \cdots x_n, \bar{x}\}.$$



# Error in Polynomial Interpolation, II

- Given the above, we have

**Theorem 5.1.16.**

If $f$ has an $(n+1)$st derivative, then for any $\bar{x}$ there is a $\xi \in [x_m, x_M]$ such that
$$f(\bar{x}) - F_{01\cdots n}(\bar{x}) = \frac{\omega(\bar{x}) f^{(n+1)}(\xi)}{(n+1)!}, \tag{5.1.23}$$
where
$$\omega(x) = (x - x_0)(x - x_1) \cdots (x - x_n). \tag{5.1.24}$$

PROOF. Consider the following function
$$G(x) = f(x) - F_{01\cdots n}(x) - K\omega(x)$$
$G(x_i) = 0, 0 \le i \le n$, since $F_{01\cdots n}(x)$ is a polynomial interpolation and by the defintion of $\omega$. We also set $G(\bar{x}) = 0$. Thus, $G(x)$ has $n+2$ zeros in $[x_m, x_M]$. By Rolle's theorem, $G'(x)$ has at least $n+1$ zeros in $[x_m, x_M]$. And, $G''(x)$ has $n$ zeros in the same interval, and so on. And finally, $G^{(n+1)}$ has at least one zero $\xi \in [x_m, x_M]$.
Since $F_{01\cdots n}(x)$ is a polynomial of order $n$, $F^{(n+1)}(x) = 0$.

# Error in Polynomial Interpolation, III

- Thus, we have
$$G^{(n+1)}(\xi) = f^{(n+1)}(\xi) - K(n+1)! = 0$$
Thus $K = \dfrac{f^{(n+1)}(\xi)}{(n+1)!}$.
And this proves the theorem. $\square$
- Note
  - The order of $\omega(\bar{x})$ increases with $n$
  - If the derivatives of $f$ is bounded in $[x_m, x_M]$, i.e., there is an integer $k$ and a $C \in \mathbb{R}$, $|f^{(j)}(x)| \leq C$ for $j > k$, then $f(x) - F_{01\cdots n}(x) \to 0$ as $n \to \infty$.
- In general, the error of polynomiar interpolation does not uniformly decrease as $n$ increases.
  - Example $f(x) = \sqrt{x}$.
  - If $f$ has break points, where the derivatives cannot be defined.



# Hermite Interpolation

- Suppose the at each $x_i$ of the support abscissas not only the value of the support ordinates, $y_i$, $0 \leq i \leq m$, are known but also the derivatives, $y_i^{(k)}$, $0 \leq k \leq n_i$, up to $n_i$th order are also known. The Hermite interpolation problem is to find a polynomial, $F$, of degree not greater than $n = (\sum_i (n_i + 1)) - 1$ such that

$$F^{(k)}(x_i) = y_i^{(k)}, 0 \leq i \leq m, 0 \leq k \leq n_i. \tag{5.1.25}$$

**Example 5.1.17.**

To find a polynomial of degree not greater than 4 such that $F(0) = 0$, $F'(0) = 0$, $F(1) = 0$, $F(2) = 1$, $F(3) = 1$.

- The support abscissases are $\{x_0, x_1, x_2, x_3\} = \{0, 1, 2, 3\}$.
- The support ordinates are $\{y_0, y_0', y_1, y_2, y_3\}$.
- Note that there are 5 conditions and thus a polynomial of order not greater than 4 can be uniquely determined.

## Hermite Interpolation, II

- Assume $F = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$, then we have

$$\begin{aligned} a_0 &= 0 \\ a_1 &= 0 \\ a_0 + a_1 + a_2 + a_3 + a_4 &= 0 \\ a_0 + a_1 \cdot 2 + a_2 \cdot 4 + a_3 \cdot 8 + a_4 \cdot 16 &= 1 \\ a_0 + a_1 \cdot 3 + a_2 \cdot 9 + a_3 \cdot 27 + a_4 \cdot 81 &= 1 \end{aligned}$$

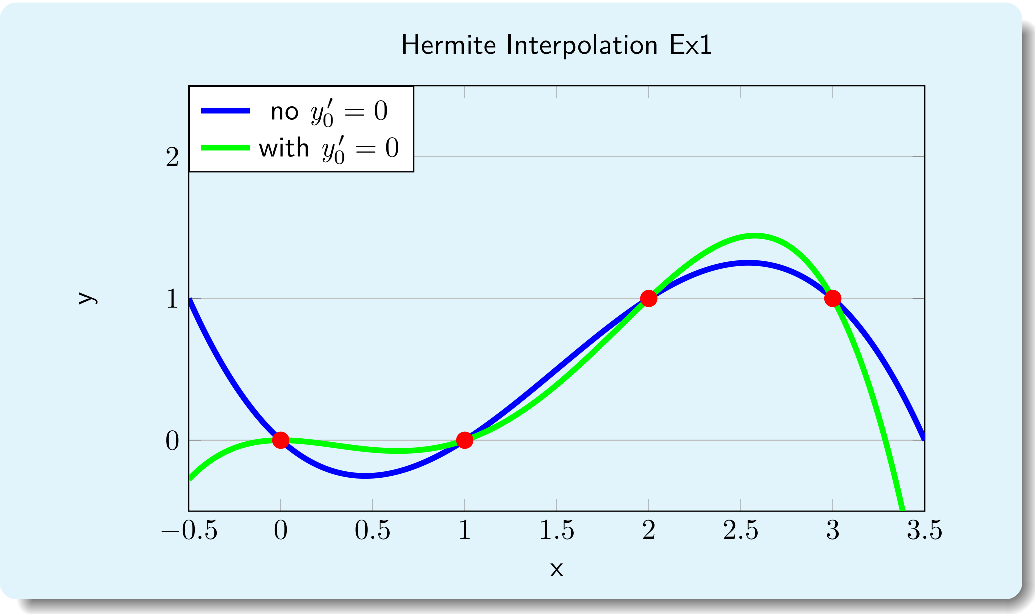
This has the following soluton:

$$a_0 = 0, a_1 = 0, a_2 = -\frac{23}{36}, a_3 = \frac{5}{6}, a_4 = -\frac{7}{36}.$$

And $F = -\frac{23}{36}x^2 + \frac{5}{6}x^3 - \frac{7}{36}x^4$



## Hermite Interpolation, III




- Note that difference adding $y_0' = 0$ to the support points.

# Hermite Interpolation, IV

- Interpolation with derivative support ordinates can also be done using Newton's interpolation formula (5.1.21).

$$
\begin{aligned}
F(x) &= F_{01\cdots n}(x) \\
&= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\
&\qquad + f[x_0, x_1, \cdots, x_n](x - x_0)\cdots(x - x_{n-1})
\end{aligned}
$$

- In this example define support abscissas and codinates as
Support abscissas $= \{x_i\} = \{0, 0, 1, 2, 3\}$,
Support ordinates $= \{y_i\} = \{0, 0, 0, 1, 1\}$,
Thus,

$$
\begin{aligned}
f[x_0] &= y_0 = 0 \\
f[x_0, x_1] &= y_0' = 0 \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{0 - 0}{1} = 0 \\
f[x_0, x_1, x_2, x_3] &= \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} = \frac{1}{4} \\
f[x_0, x_1, x_2, x_3, x_4] &= \frac{f[x_1, x_2, x_3, x_4] - f[x_0, x_1, x_2, x_3]}{x_4 - x_0} = -\frac{7}{36}.
\end{aligned}
$$



# Summary

- Interpolation problems
- Interpolation by polynomials
- Lagrange interpolation formula
- Neville's algorithm
    - Recursive and nonrecursive forms
- Newton's interpolation formula
    - Divided differences
- Errors
- Hermite interpolation