Given a set of discrete values, how can we estimate other values between these data

The method that we will use is called polynomial interpolation.

We assume the data we had are from the evaluation of a smooth function. We may be able to use a polynomial \( p(x) \) to approximate this function, at least locally.

A condition: the polynomial \( p(x) \) takes the given values at the given points (nodes), i.e., \( p(x_i) = y_i \) with \( 0 \leq i \leq n \). The polynomial is said to interpolate the table, since we do not know the function.
Note that all the points are passed through by the curve.
We do not know the original function, the interpolation may not be accurate outside the set of data points.
A polynomial of degree 0, a constant function, interpolates one set of data.

If we have two sets of data, we can have an interpolating polynomial of degree 1, a linear function:

\[ p(x) = \left( \frac{x - x_1}{x_0 - x_1} \right)y_0 + \left( \frac{x - x_0}{x_1 - x_0} \right)y_1 \]

\[ = y_0 + \left( \frac{y_1 - y_0}{x_1 - x_0} \right)(x - x_0) \]

Review carefully if the condition is satisfied.

Interpolating polynomials can be written in several forms, the most well known ones are the Lagrange form and Newton form. Each has some advantages.
For a set of fixed nodes \( x_0, x_1, \ldots, x_n \), the cardinal functions, \( l_0, l_1, \ldots, l_n \), are defined as

\[
l_i(x_j) = \delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}
\]

We can interpolate any function \( f(x) \) by the Lagrange form of the interpolating polynomial of degree \( \leq n \)

\[
p_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i)
\]

Note that \( l_i(x) \) is of order \( n \), so \( p_n(x) \) is of order \( \leq n \), and

\[
p_n(x_j) = \sum_{i=1}^{n} l_i(x_j) f(x_i) = l_j(x_j) f(x_j) = f(x_j)
\]

The (point exact) interpolation condition is satisfied
Cardinal Functions

The cardinal function is

\[ l_i(x) = \prod_{j \neq i, j=0}^{n} \left( \frac{x - x_j}{x_i - x_j} \right) \quad (0 \leq i \leq n) \]

What it looks like

\[ l_i(x) = \left( \frac{x - x_0}{x_i - x_0} \right) \left( \frac{x - x_1}{x_i - x_1} \right) \ldots \]
\[ \left( \frac{x - x_{i-1}}{x_i - x_{i-1}} \right) \left( \frac{x - x_{i+1}}{x_i - x_{i+1}} \right) \ldots \]
\[ \left( \frac{x - x_n}{x_i - x_n} \right) \]

Note that \( l_i(x_j) = 0 \), for \( i \neq j \)

and \( l_i(x_i) = 1 \)
Five Lagrange Basis Polynomials

These are the 5 Lagrange basis polynomials for $N=5$
Representation by Lagrange Polynomials

Five weighted polynomials and their sum (red line) for a set of 5 random samples (red points)
Step by Step Construction

For any table of data, we can construct a Lagrange interpolating polynomial. Its evaluation is a little bit costly, but we can always do that. The existence of the interpolating polynomial is guaranteed.

Can we construct the interpolating polynomial step by step, or if we discover some new data, can we add those data to the existing interpolating polynomial to make the interpolation more accurate?

We can use Newton form of the interpolating polynomial.

Let $p_k(x)$ be an interpolating polynomial for the data set $\{(x_i,y_i)\}$ with $0 \leq i \leq k$ such that $p_k(x_i) = y_i$. 
We want to add another data \((x_{k+1}, y_{k+1})\) to have a new interpolating polynomial \(p_{k+1}(x)\) such that \(p_{k+1}(x_i) = y_i\) for \(0 \leq i \leq (k + 1)\).

Let

\[
p_{k+1}(x) = p_k(x) + c(x - x_0)(x - x_1) \cdots (x - x_k)
\]

where \(c\) is an undetermined constant.

Since \(p_{k+1}(x_{k+1}) = y_{k+1}\), we have

\[
p_k(x_{k+1}) + c(x_{k+1} - x_0)(x_{k+1} - x_1) \cdots (x_{k+1} - x_k) = y_{k+1}
\]

We can solve this equation for \(c\), with the condition that \(x_0, x_1, \ldots, x_{k+1}\) are all distinct.

\[
c = \frac{y_{k+1} - p_k(x_{k+1})}{(x_{k+1} - x_0)(x_{k+1} - x_1) \cdots (x_{k+1} - x_k)}
\]
Newton Polynomial Interpolation

Newton Interpolation: Uniform spaced nodes

- Samples
- Truth
- Interpolation
Uniqueness of Polynomial

Is the interpolating polynomial unique?

If \( p \) and \( q \) are interpolating polynomials for the data set \( \{(x_i, y_i)\} \) for \( 0 \leq i \leq n \) such that \( p(x_i) = q(x_i) = y_i \)

Then the polynomial \( r(x) = p(x) - q(x) \) of degree at most \( n \) is zero at \( x_0, x_1, \ldots, x_n \). Note that a polynomial of degree \( n \) can have at most \( n \) roots, we must have \( r(x) = 0 \), or \( p - q = 0 \). Hence \( p = q \)

The interpolating polynomial is unique

It may be written in different forms
An Example

Find the interpolating polynomial for this table

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>-5</td>
<td>-3</td>
<td>-15</td>
</tr>
</tbody>
</table>

Lagrange form

\[ l_0(x) = \frac{(x - 1)(x + 1)}{(0 - 1)(0 + 1)} = -(x - 1)(x + 1) \]

\[ l_1(x) = \frac{(x - 0)(x + 1)}{(1 - 0)(1 + 1)} = \frac{1}{2} x(x + 1) \]

\[ l_2(x) = \frac{(x - 0)(x - 1)}{(-1 - 0)(-1 - 1)} = \frac{1}{2} x(x - 1) \]

The interpolating polynomial is

\[ p_2(x) = 5(x - 1)(x + 1) - \frac{3}{2} x(x + 1) - \frac{15}{2} x(x - 1) \]
The zeroth order polynomial is

\[ p_0(x) = -5 \]

Let the 1\textsuperscript{st} order interpolating polynomial be

\[ p_1(x) = p_0 + c(x - x_0) = -5 + c(x - 0) \]

We want \( p_1(x_1) = -3 \), hence \(-5 + c(1 - 0) = -3\), we have \( c = 2 \), it follows that

\[ p_1(x) = -5 + 2x \]

Let the 2\textsuperscript{nd} order interpolating polynomial be

\[ p_2(x) = p_1(x) + c(x - x_0)(x - x_1) \]

Put \( p_2(-1) = -15 \), i.e., \(-5+2(-1) + c(-1 - 0)(-1 - 1) = -15\). We have \( c = -4 \).

The Newton form of the interpolating polynomial is

\[ p_2(x) = -5 + 2x - 4x(x - 1) \]
For easy programming and efficient computation, we can write Newton form of the interpolating polynomial in **nested form**

\[
p(x) = a_0 + a_1 [(x - x_0)] + a_2 [(x - x_0)(x - x_1)] + a_3 [(x - x_0)(x - x_1)(x - x_2)] + \cdots + a_n [(x - x_0)(x - x_1) \cdots (x - x_{n-1})]
\]

Or, using standard product notations as

\[
p(x) = a_0 + \sum_{i=1}^{n} a_i \left[ \prod_{j=1}^{i-1} (x - x_j) \right]
\]

Using successive factorization, the nested form is

\[
p(x) = a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + \cdots + (x - x_{n-1})a_n)) \cdots \\
= (\cdots ((a_n(x - x_{n-1}) + a_{n-1})(x - x_{n-2}) + a_{n-2}) \cdots)(x - x_0) + a_0
\]
Computation Procedure

To evaluate $p(x)$ for a given $x$, we start from the innermost parentheses, forming successively some intermediate quantities

$$v_0 = a_n$$
$$v_1 = v_0(x - x_{n-1}) + a_{n-1}$$
$$v_2 = v_1(x - x_{n-2}) + a_{n-2}$$
$$\vdots$$
$$v_i = v_{i-1}(x - x_{n-i}) + a_{n-i}$$
$$\vdots$$
$$v_n = v_{n-1}(x - x_0) + a_0$$

A pseudocode is

```plaintext
real array (a_i)_{0:n}, (x_i)_{0:n}
integer i, n
real x, v
v ← a_n
for i = n - 1 to 0 step -1 do
    v ← v(x - x_i) + a_i
end for
```
The coefficients $a_i$ in Newton form of the interpolating polynomial need to be computed. A notation is introduced to facilitate such computation

$$a_k = f[x_0, x_1, \ldots, x_k]$$

which is called the **divided difference of order** $k$ for $f$.

Newton form interpolating polynomial is

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

Or written in a compact form

$$p_n(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1} (x - x_j)$$

with the convention

$$\prod_{j=0}^{-1} (x - x_j) = 1$$
Computing Coefficients $a_i$

We want $p_n(x_i) = f(x_i)$. So we have

\[ f(x_0) = a_0 \]
\[ f(x_1) = a_0 + a_1(x_1 - x_0) \]
\[ f(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \]

... ... ...

The solution of this system is

\[ a_0 = f(x_0) \]

\[ a_1 = \frac{f(x_1) - a_0}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \]

The divided difference of order 1 is

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

Note that $f[x_0, x_1, \ldots, x_k]$ is the coefficient of $x^k$ in the polynomial $p_k$ of degree $\leq k$
In general, we have
\[
f[x_0, x_1, \ldots, x_k] = \frac{f(x_k) - \sum_{i=0}^{k-1} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x_k - x_j)}{\prod_{j=0}^{k-1} (x_k - x_j)}
\]

Computational algorithm

• Set \( f[x_0] = f(x_0) \)

• For \( k = 1, 2, \ldots, n \), compute
  \( f[x_0, x_1, \ldots, x_k] \) using the above equation
Recursive Formula

The divided difference has a recursive formula

\[
    f[x_0, x_1, \ldots, x_k] = \frac{f[x_1, x_2, \ldots, x_k] - f[x_0, x_1, \ldots, x_{k-1}]}{x_k - x_0}
\]

Proof:

\(f[x_0, x_1, \ldots, x_k]\) is the coefficient of \(x^k\) in the polynomial \(p_k\) of degree \(\leq k\), which interpolates \(f\) at \(x_0, x_1, \ldots, x_k\)

\(f[x_1, x_2, \ldots, x_k]\) is the coefficient of \(x^{k-1}\) in the polynomial \(q_{k-1}\) of degree \(\leq (k - 1)\), which interpolates \(f\) at \(x_1, x_2, \ldots, x_k\)

\(f[x_0, x_1, \ldots, x_{k-1}]\) is the coefficient of \(x^{k-1}\) in the polynomial \(p_{k-1}\) of degree \(\leq (k - 1)\), which interpolates \(f\) at \(x_0, x_1, \ldots, x_{k-1}\)
Recursive Formula - Proof

We have

\[ p_k(x) = q_{k-1}(x) + \frac{x - x_k}{x_k - x_0} [q_{k-1}(x) - p_{k-1}(x)] \]

To prove this identity, it suffices to show that it holds at \((k + 1)\) different points, since the left-hand side and the right-hand side are polynomials of degree \(\leq k\). Note that the left-hand side is \(p_k(x_i) = f(x_i)\) for \(i = 0, 1, \ldots, k\).

Check the right-hand side at point \(x_0\)

\[
q_{k-1}(x_0) + \frac{x_0 - x_k}{x_k - x_0} [q_{k-1}(x_0) - p_{k-1}(x_0)]
= q_{k-1}(x_0) - [q_{k-1}(x_0) - p_{k-1}(x_0)]
= p_{k-1}(x_0) = f(x_0)
\]
Check for points $1 \leq i \leq (k - 1)$,

$$q_{k-1}(x_i) + \frac{x_i - x_k}{x_k - x_0} [q_{k-1}(x_i) - p_{k-1}(x_i)]$$

$$= f(x_i) + \frac{x_i - x_k}{x_k - x_0} [f(x_i) - f(x_i)] = f(x_i)$$

Check the right-hand side at point $x_k$

$$q_{k-1}(x_k) + \frac{x_k - x_k}{x_k - x_0} [q_{k-1}(x_k) - p_{k-1}(x_k)]$$

$$= q_{k-1}(x_k) = f(x_k)$$

Hence the said identity holds

We take the coefficients of $x^k$ on both sides, which yields the desired recursive formula
Invariance Theorem

The divided difference $f[x_0, x_1, \ldots, x_k]$ is invariant under all permutations of the arguments $x_0, x_1, \ldots, x_k$

This is because $f[x_0, x_1, \ldots, x_k]$ is the coefficient of $x^k$ of the polynomial $p_k(x)$ of degree $\leq k$ that interpolates $f$ at $x_0, x_1, \ldots, x_k$. $f[x_1, x_0, \ldots, x_k]$ is the coefficient of $x^k$ of the polynomial $p_k(x)$ of degree $\leq k$ that interpolates $f$ at $x_1, x_0, \ldots, x_k$. These two polynomials are the same.

The generic recursive formula is

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

$$f[x_{i+1}, x_{i+2}, \ldots, x_j] - f[x_i, x_{i+1}, \ldots, x_{j-1}]$$

$$\frac{x_j - x_i}{x_j - x_i}$$
We can construct a divided difference table for \( f \) to facilitate computation of the coefficients of the interpolating polynomial.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f[\ ] )</th>
<th>( f[\ , \ ] )</th>
<th>( f[\ , \ , \ ] )</th>
<th>( f[\ , \ , \ , \ ] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_0 )</td>
<td>( f[x_0] )</td>
<td>( f[x_0, x_1] )</td>
<td>( f[x_0, x_1, x_2] )</td>
<td>( f[x_0, x_1, x_2, x_3] )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( f[x_1] )</td>
<td>( f[x_1, x_2] )</td>
<td>( f[x_1, x_2, x_3] )</td>
<td>( f[x_1, x_2, x_3] )</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( f[x_2] )</td>
<td>( f[x_2, x_3] )</td>
<td>( f[x_2, x_3] )</td>
<td>( f[x_2, x_3] )</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>( f[x_3] )</td>
<td>( f[x_3] )</td>
<td>( f[x_3] )</td>
<td>( f[x_3] )</td>
</tr>
</tbody>
</table>

The coefficients along the top diagonal are the ones needed to form the Newton form of the interpolating polynomial.
## Divided Difference Table

<table>
<thead>
<tr>
<th>i</th>
<th>$x_i$</th>
<th>$f[x_i]$</th>
<th>First order differences</th>
<th>Second order differences</th>
<th>Third order differences</th>
<th>Fourth order differences</th>
<th>Fifth order differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_0$</td>
<td>$f[x_0]$</td>
<td>$f[x_0, x_1]$</td>
<td>$f[x_0, x_1, x_2]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$f[x_1]$</td>
<td></td>
<td>$f[x_1, x_2]$</td>
<td>$f[x_1, x_2, x_3]$</td>
<td>$f[x_0, x_1, x_2, x_3]$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$x_2$</td>
<td>$f[x_2]$</td>
<td></td>
<td></td>
<td>$f[x_1, x_2, x_3]$</td>
<td></td>
<td>$f[x_0, x_1, x_2, x_3, x_4]$</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$f[x_3]$</td>
<td></td>
<td></td>
<td></td>
<td>$f[x_1, x_2, x_3, x_4]$</td>
<td>$f[x_0, x_1, x_2, x_3, x_4, x_5]$</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$f[x_4]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$x_5$</td>
<td>$f[x_5]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A pseudocode for computing divided difference is

```
real array (a_{ij})_{0:n \times 0:n}, (x_i)_{0:n}
integer i, j, n
for i = 0 to n do
    a_{i0} ← f(x_i)
end for
for j = 1 to n do
    for i = 0 to n – j do
        a_{ij} ← (a_{i+1,j-1} – a_{i,j-1})/(x_{i+j} – x_i)
    end for
end for
```

This algorithm computes and stores all components of the divided difference. The coefficients of the Newton interpolating polynomial are stored in the first row of the array (a_{ij})_{0:n \times 0:n}, i.e., in a(0 : n, 0)
Computing the Coefficients only

If we compute divided difference only for constructing Newton interpolating polynomial, there is no need to store the unnecessary divided difference terms. (But they will be computed, used, and discarded)

real array $(a_i)_{0:n}$, $(x_i)_{0:n}$
integer $i$, $j$, $n$
for $i = 0$ to $n$ do
    $a_i \leftarrow f(x_i)$
end for
for $j = 1$ to $n$ do
    for $i = n$ to $j$ step $-1$ do
        $a_i \leftarrow (a_i - a_{i-1})/(x_i - x_{i-j})$
    end for
end for

The two algorithms assume the same computational cost
Memory Allocations

Here we show how the memory is occupied and updated in computing coefficients of Newton interpolating polynomial

<table>
<thead>
<tr>
<th></th>
<th>$f[x_0]$</th>
<th>$f[x_1]$</th>
<th>$f[x_2]$</th>
<th>$f[x_3]$</th>
<th>$f[x_4]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5th</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computation must be done backward to avoid erasing needed memory locations
# Divided Difference Table

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x_i )</th>
<th>( f[x_i] )</th>
<th>First order differences</th>
<th>Second order differences</th>
<th>Third order differences</th>
<th>Fourth order differences</th>
<th>Fifth order differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_0 )</td>
<td>( f[x_0] )</td>
<td>( f[x_0, x_1] )</td>
<td>( f[x_0, x_1, x_2] )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( x_1 )</td>
<td>( f[x_1] )</td>
<td></td>
<td></td>
<td>( f[x_1, x_2, x_3] )</td>
<td>( f[x_0, x_1, x_2, x_3] )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( x_2 )</td>
<td>( f[x_2] )</td>
<td>( f[x_1, x_2] )</td>
<td>( f[x_1, x_2, x_3] )</td>
<td></td>
<td></td>
<td>( f[x_1, x_2, x_3, x_4, x_5] )</td>
</tr>
<tr>
<td>3</td>
<td>( x_3 )</td>
<td>( f[x_3] )</td>
<td>( f[x_2, x_3] )</td>
<td>( f[x_2, x_3, x_4] )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( x_4 )</td>
<td>( f[x_4] )</td>
<td>( f[x_3, x_4] )</td>
<td></td>
<td>( f[x_2, x_3, x_4, x_5] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( x_5 )</td>
<td>( f[x_5] )</td>
<td>( f[x_4, x_5] )</td>
<td></td>
<td></td>
<td>( f[x_3, x_4, x_5] )</td>
<td></td>
</tr>
</tbody>
</table>
Newton Polynomial

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$y_i = f(x_i)$</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>27</td>
<td>64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>$x_i$</th>
<th>$f[x_i]$</th>
<th>1$^{st}$ order differences</th>
<th>2$^{nd}$ order differences</th>
<th>3$^{rd}$ order differences</th>
<th>4$^{th}$ order differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\frac{1 - 0}{1 - 0}$ = 1</td>
<td>$\frac{7 - 1}{2 - 0}$ = 3</td>
<td>$\frac{6 - 3}{3 - 0}$ = 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\frac{8 - 1}{2 - 1}$ = 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>8</td>
<td></td>
<td>$\frac{19 - 7}{3 - 1}$ = 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td></td>
<td></td>
<td>$\frac{37 - 19}{4 - 2}$ = 9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td>$\frac{64 - 27}{4 - 3}$ = 9</td>
</tr>
</tbody>
</table>
**Inverse Interpolation**

It is also possible to use a polynomial to approximate the inverse of a function \( y = f(x) \). Given a table

<table>
<thead>
<tr>
<th>( y )</th>
<th>( y_0 )</th>
<th>( y_1 )</th>
<th>( \cdots )</th>
<th>( y_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( x_0 )</td>
<td>( x_1 )</td>
<td>( \cdots )</td>
<td>( x_n )</td>
</tr>
</tbody>
</table>

An interpolation polynomial

\[
p(y) = \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1} (y - y_j)
\]

Can be constructed such that \( p(y_i) = x_i \). This interpolating polynomial is useful to find the approximate location of a root of a function \( f(x) \). E.g., there is a root in \([4.0,5.0]\) for this table

<table>
<thead>
<tr>
<th>( y )</th>
<th>-0.579</th>
<th>-0.363</th>
<th>-0.185</th>
<th>-0.034</th>
<th>0.097</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
<td>4.0</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Neville’s Algorithm

Neville proposed a different scheme to construct interpolation polynomial step by step. Start with zero degree polynomials $P_i(x) = f(x_i)$, we construct higher degree interpolation polynomials by the recurrence relation

$$S_{ij}(x) = \left( \frac{x - x_{i-j}}{x_i - x_{i-j}} \right) S_{i,j-1}(x)$$

$$+ \left( \frac{x_i - x}{x_i - x_{i-j}} \right) S_{i-1,j-1}(x)$$

With $S_{i0}(x) = P_i(x) = f(x_i)$. The relation table can be written as

<table>
<thead>
<tr>
<th>x_i</th>
<th>S_{i0}(x)</th>
<th>S_{i1}(x)</th>
<th>S_{i2}(x)</th>
<th>S_{i3}(x)</th>
<th>S_{i4}(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_0</td>
<td>S_{00}(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_1</td>
<td>S_{10}(x)</td>
<td>S_{11}(x)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_2</td>
<td>S_{20}(x)</td>
<td>S_{21}(x)</td>
<td>S_{22}(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_3</td>
<td>S_{30}(x)</td>
<td>S_{31}(x)</td>
<td>S_{32}(x)</td>
<td>S_{33}(x)</td>
<td></td>
</tr>
<tr>
<td>x_4</td>
<td>S_{40}(x)</td>
<td>S_{41}(x)</td>
<td>S_{42}(x)</td>
<td>S_{43}(x)</td>
<td>S_{44}(x)</td>
</tr>
</tbody>
</table>
Redefine constant polynomials as \( P_i^{(0)}(x) = y_i \) for \( 0 \leq i \leq n \),

We can define higher order polynomial as

\[
P_i^{(j)}(x) = \left( \frac{x - x_{i-j}}{x_i - x_{i-j}} \right) P_i^{(j-1)}(x)
\]

\[
+ \left( \frac{x_i - x}{x_i - x_{i-j}} \right) P_{i-1}^{(j-1)}(x)
\]

The range of \( j \) is \( 1 \leq j \leq n \) and that of \( i \) is \( j \leq i \leq n \)

The interpolation properties of these polynomials are:

The polynomial \( P_i^{(j)} \) defined above interpolate as follows (see p. 153 for a proof)

\[
P_i^{(j)}(x_k) = y_k \quad (0 \leq i - j \leq k \leq i \leq n)
\]
Higher Dimensional Interpolation

It is possible to define interpolation polynomials of several variables. The tensor-product interpolation is used on rectangular domain \([a, b] \times [\alpha, \beta]\). Select \(n\) nodes in \([a, b]\) and define the Lagrange polynomials as

\[
l_i(x) = \prod_{j \neq i, j=1}^{n} \frac{x - x_j}{x_i - x_j} \quad (1 \leq i \leq n).
\]

Select \(m\) nodes in \([\alpha, \beta]\) and define

\[
h_i(y) = \prod_{j \neq i, j=1}^{n} \frac{y - y_j}{y_i - y_j} \quad (1 \leq i \leq m).
\]

Then function

\[
P(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} f(x_i, y_j) l_i(x) h_j(y)
\]

Interpolates a two dimensional table with data \((x_i, y_j, f(x_i, y_j))\)
2D Interpolation
Errors of Interpolation I

If \( p \) is the polynomial of degree at most \( n \) that interpolates \( f \) at the \( n+1 \) distinct nodes \( x_0, x_1, \ldots, x_n \) belonging to an interval \([a,b]\) and if \( f^{(n+1)} \) is continuous for each \( x \) in \([a,b]\), there is a \( z \) in \((a, b)\) for which

\[
f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(z) \prod_{i=0}^{n} (x - x_i)
\]

What kind of function may have a small interpolation error?
Errors of Interpolation Proof

The error formula is correct if \( x \) is one of the nodes \( x_i \)

Assume \( x \) is not a node and is fixed, we define

\[
    w(t) = \prod_{i=0}^{n} (t - x_i)
\]

\[
    c = \frac{f(x) - p(x)}{w(x)} \quad \text{(This is a constant)}
\]

\[
    \phi(t) = f(t) - p(t) - cw(t)
\]

\( c \) is well defined as \( w(x) \neq 0 \). \( \phi \) has \((n+2)\) zeros at \( x_0, x_1, ..., x_n \) and \( x \). According to Rolle’s Theorem, there is a root for \( \phi'(t) \) between any two roots of \( \phi(t) \). Thus \( \phi'(t) \) has at least \((n+1)\) zeros. Similarly \( \phi''(t) \) has at least \( n \) zeros, ..., and \( \phi^{(n+1)}(t) \) has at least one zero.
Errors of Interpolation Proof

Let \( z \) be a zero of \( \varphi^{(n+1)}(t) \), we have

\[
0 = \varphi^{(n+1)}(z) = f^{(n+1)}(z) - p^{(n+1)}(z) - c w^{(n+1)}(z)
\]

Here \( p^{(n+1)}(z) = 0 \) because \( p \) is a polynomial of degree \( n \) or less. Also

\[
w^{(n+1)}(z) = (n + 1)!
\]

Since

\[
w(t) = \prod_{i=0}^{n} (t - x_i) = t^{n+1} + \text{(lower order terms in } t)\]

So, we have

\[
0 = f^{(n+1)}(z) - c(n + 1)! = f^{(n+1)}(z) - \frac{(n+1)!}{w(x)} [f(x) - p(x)]
\]
Errors of Interpolation II

Let $f$ be a function such that $f^{(n+1)}$ is continuous on $[a,b]$ and satisfies

$$|f^{(n+1)}(x)| \leq M.$$

Let $p$ be the polynomial of degree at most $n$ that interpolates $f$ at the $n+1$ equally spaced nodes $x_0, x_1, \ldots, x_n$ in $[a,b]$. Then on $[a,b]$,

$$|f(x) - p(x)| \leq \frac{1}{4(n+1)} Mh^{n+1}$$

Where $h=(b-a)/n$ is the spacing between the nodes.
Computing First Derivative

The first derivative can be approximated as

$$f'(x) \approx \frac{1}{h} [f(x + h) - f(x)] \quad (1)$$

For accurate approximation, $h$ should be small. Thus $f(x + h)$ and $f(x)$ are close to each other. This may cause loss of significant digits in finite precision computation.

Using Taylor’s theorem, we have

$$f(x + h) = f(x) + hf'(x) + \frac{1}{2} h^2 f''(\xi)$$

For $\xi$ between $x$ and $x + h$. It follows that

$$f'(x) = \frac{1}{h} [f(x + h) - f(x)] - \frac{1}{2} hf''(\xi)$$

The approximation error of (1) is $- \frac{1}{2} hf''(\xi)$, or of order $O(h)$. This is a first order (or sided) approximation of first derivative. The error goes to 0 as fast as $h \to 0$. 
Higher Order Approximation

It is desirable to have some higher order (faster) approximation schemes

\[
\begin{align*}
    f(x + h) &= f(x) + hf'(x) + \frac{1}{2!} h^2 f''(x) \\
               &\quad + \frac{1}{3!} h^3 f'''(x) + \frac{1}{4!} h^4 f^{(4)}(x) + \cdots \\
    f(x - h) &= f(x) - hf'(x) + \frac{1}{2!} h^2 f''(x) \\
               &\quad - \frac{1}{3!} h^3 f'''(x) + \frac{1}{4!} h^4 f^{(4)}(x) - \cdots
\end{align*}
\]

Subtracting these two equations, we have

\[
f(x + h) - f(x - h) = 2hf'(x) \\
+ \frac{2}{3!} h^3 f'''(x) + \frac{2}{5!} h^5 f^{(5)}(x) + \cdots
\]
It follows that
\[
f'(x) = \frac{1}{2h} \left[ f(x + h) - f(x - h) \right]
\]
\[- \frac{h^2}{3!} f'''(x) - \frac{h^4}{5!} f^{(5)}(x) - \cdots\]

After dropping the higher order terms, we have a second order approximation formula as
\[
f'(x) \approx \frac{1}{2h} \left[ f(x + h) - f(x - h) \right]
\]

The leading truncated terms of this approximation scheme is $-\frac{h^2}{6} f'''(x)$. Hence the approximation is of $O(h^2)$. The approximation error goes to 0 as fast as $h^2 \to 0$. The exact truncation error is
\[
-\frac{1}{6} h^2 \left[ \frac{f'''(\xi_1) + f'''(\xi_2)}{2} \right] = -\frac{1}{6} h^2 f'''(\xi)
\]
2nd Order Approximation
Richardson Extrapolation (I)

First derivative can be approximated as
\[ f'(x) = \frac{1}{2h} [f(x + h) - f(x - h)] \]
\[ + a_2 h^2 + a_4 h^4 + a_6 h^6 + \cdots \]

In which the constants \( a_2, a_4, \ldots \) depend on the higher order derivatives of \( f \) and the value of \( x \), (but) not on \( h \). When such information is available, it is possible to construct much more accurate approximation schemes.

Define a function
\[ \psi(h) = \frac{1}{2h} [f(x + h) - f(x - h)] \]

Which is an approximation to \( f''(x) \) with error of order \( O(h^2) \). This approximation becomes accurate as \( h \to 0 \). So we can study the quantity \( \lim_{h \to 0} \psi(h) \).
Richardson extrapolation estimates the value of $\psi(0)$ from some computed values of $\psi(h)$ near 0

$$\psi(h) = f'(x) - a_2 h^2 - a_4 h^4 - a_6 h^6 - \cdots$$

$$\psi\left(\frac{h}{2}\right) = f'(x) - a_2 \left(\frac{h}{2}\right)^2 - a_4 \left(\frac{h}{2}\right)^4 - a_6 \left(\frac{h}{2}\right)^6 - \cdots$$

Multiply the 2nd equation by 4 and subtract it from the 1st equation

$$\psi(h) - 4\psi\left(\frac{h}{2}\right) = -3 f'(x) - \frac{3}{4} a_4 h^4 - \frac{15}{16} a_6 h^6 - \cdots$$

Hence

$$\psi\left(\frac{h}{2}\right) + \frac{1}{3} \left[ \psi\left(\frac{h}{2}\right) - \psi(h) \right] = f'(x) + \frac{a_4}{4} h^4 + \frac{5}{16} a_6 h^6 + \cdots$$

$f'(x)$ can be computed as accurate as $O(h^4)$
A general Richardson extrapolation form

$$\psi(h) = L - \sum_{k=1}^{\infty} a_{2k} h^{2k}$$

Where we assume that $\psi(h)$ is computable for any $h > 0$ and we want to approximate $L$ as accurately as possible

Choose a special sequence define

$$D(n,0) = \psi\left(\frac{h}{2^n}\right) \quad (n \geq 0)$$

Then, we have

$$D(n,0) = L + \sum_{k=1}^{\infty} A(k,0) \left(\frac{h}{2^n}\right)^{2k}$$

With $A(k,0) = -a_{2k}$. $D(n,0)$ is a rough approximate of $L = \lim_{x \to 0} \psi(x)$
Richardson Theorem

The extrapolation formula is

\[ D(n,m) = \frac{4^m}{4^m - 1} D(n,m - 1) - \frac{1}{4^m - 1} D(n - 1, m - 1) \quad (1 \leq m \leq n) \]

Richardson Extrapolation Theorem:

\[ D(n,m) = L + \sum_{k=m+1}^{\infty} A(k,m) \left( \frac{h}{2^n} \right)^{2k} \]

For \( 0 \leq m \leq n \)

The proof of this theorem is based on induction on \( m \), see p. 175 of the Book. Proof will not be given in class.

Not that \( D(n,m) \) approximates \( L \) at the order of \( O(h^{2m}) \). The convergence rate is fast.
Richardson extrapolation computational procedure:

1.) write a procedure to compute $\psi(h)$

2.) decide on suitable values for $n$ and $h$

3.) for $i = 0, 1, \ldots, n$, compute

$$D(i,0) = \psi(h / 2^i)$$

4.) for $0 \leq i \leq j \leq n$, compute

$$D(i, j) = D(i, j - 1) + (4^j - 1)^{-1}[D(i, j - 1) - D(i - 1, j - 1)]$$
Using Interpolation Polynomial

We can approximate the function \( f(x) \) by a polynomial \( p_n(x) \) of order \( n \), such that \( p_n(x) \approx f(x) \)

To compute \( f'(x) \), we use the approximation \( f'(x) \approx p'_n(x) \)

Higher order polynomials are avoided because of oscillation

Let \( p \) interpolates \( f \) at two points, \( x_0 \) and \( x_1 \)

\[
p_1(x) = f(x_0) + f[x_0, x_1](x - x_0)
\]

The first derivative of \( p_1(x) \) is

\[
p'_1(x) = f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \approx f'(x)
\]
1\textsuperscript{st} and 2\textsuperscript{nd} Order Approx.

Let $x_0 = x$ and $x_1 = x + h$, we have

$$f''(x) \approx \frac{1}{h} [f(x + h) - f(x)]$$

This is just the $O(h)$ order sided approximation formula.

Put $x_0 = x - h$ and $x_1 = x + h$, we have the $O(h^2)$ approximation scheme

$$f''(x) \approx \frac{1}{2h} [f(x + h) - f(x - h)]$$

A three point polynomial interpolation is

$$p_2(x) = f(x_0) + f[x_0, x_1] (x - x_0) + f[x_0, x_1, x_2] (x - x_0)(x - x_1)$$

We have corrected approximation

$$p'_2(x) = f[x_0, x_1] + f[x_0, x_1, x_2] (2x - x_0 - x_1)$$
Second Derivative

If we have first derivative, we can use

\[ f''(x) = \frac{1}{2h} [f'(x + h) - f'(x - h)] \]

To approximate the second derivative to \(O(h^2)\)

A direct approximation would be using Taylor expansion

\[ f(x + h) + f(x - h) = 2f(x) + h^2 f''(x) + 2 \left[ \frac{1}{4!} h^4 f^{(4)}(x) + \cdots \right] \]

Hence, we have

\[ f''(x) \approx \frac{1}{h^2} [f(x + h) - 2f(x) + f(x - h)] \]

This approximation is of \(O(h^2)\) accuracy