Interpolation

1 What is interpolation?

For a certain function $f(x)$ we know only the values $y_1 = f(x_1), \ldots, y_n = f(x_n)$. For a point $\tilde{x}$ different from $x_1, \ldots, x_n$ we would then like to approximate $f(\tilde{x})$ using the given data $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.

This means we are constructing a function $p(x)$ which passes through the given points and hopefully is close to the function $f(x)$. It turns out that it is a good idea to use polynomials as interpolating functions (later we will also consider piecewise polynomial functions).

2 Why are we interested in this?

- **Efficient evaluation of functions:** For functions like $f(x) = \sin(x)$ it is possible to find values using a series expansion (e.g. Taylor series), but this takes a lot of operations. If we need to compute $f(x)$ for many values x in an interval $[a, b]$ we can do the following:
  - pick points $x_1, \ldots, x_n$ in the interval
  - find the interpolating polynomial $p(x)$
  - Then: for any given $x \in [a, b]$ just evaluate the polynomial $p(x)$ (which is cheap) to obtain an approximation for $f(x)$

  Before the age of computers and calculators, values of functions like $\sin(x)$ were listed in tables for values x_j with a certain spacing. Then function values everywhere in between could be obtained by interpolation.

  A computer or calculator uses the same method to find values of e.g. $\sin(x)$: First an interpolating polynomial $p(x)$ for the interval $[0, \pi/2]$ was constructed and the coefficients are stored in the computer. For a given value $x \in [0, \pi/2]$, the computer just evaluates the polynomial $p(x)$ (once we know the sine function for $[0, \pi/2]$ we can find $\sin(x)$ for all x).
- **Design of curves:** For designing shapes on a computer we would like to pick a few points with the mouse, and then the computer should find a "smooth curve" which passes through the given points.
- **Tool for other algorithms:** In many cases we only have data $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ for a function $f(x)$, but we would like to compute things like
  - the integral $I = \int_a^b f(x)dx$
  - a "zero" x_* of the function where $f(x_*) = 0$
  - a derivative $f'(\tilde{x})$ at some point $\tilde{x}$.

  We can do all this by first constructing the interpolating polynomial $p(x)$. Then we can approximate I by $\int_a^b p(x)dx$. We can approximate x_* by finding a zero of the function $p(x)$. We can approximate $f'(\tilde{x})$ by evaluating $p'(\tilde{x})$.

3 Interpolation with polynomials

3.1 Basic idea

If we have two points (x_1, y_1) and (x_2, y_2) the obvious way to guess function values at other points would be to use the linear function $p(x) = c_0 + c_1 x$ passing through the two points. We can then approximate $f(\tilde{x})$ by $p(\tilde{x})$.

If we have three points we can try to find a function $p(x) = c_0 + c_1 x + c_2 x^2$ passing through all three points.

If we have n points we can try to find a function $p(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$ passing through all n points.

3.2 Existence and uniqueness

We first have to make sure that our interpolation problem always has a unique solution.

Theorem 3.1. *Assume that $x_1, \ldots, x_n$ are different from each other. Then for any $y_1, \ldots, y_n$ there exists a unique polynomial $p_{n-1}(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$ such that*

$$p(x_j) = y_j \qquad \textit{for } j = 1, \ldots, n.$$

Proof. We use induction. **Induction start:** For $n = 1$ we need to find $p_0(x) = a_0$ such that $p_0(x_1) = y_1$. Obviously this has a unique solution

$$a_0 = y_1. \tag{1}$$

Induction step: We assume that the theorem holds for n points. Therefore there exists a unique polynomial $p_{n-1}(x)$ with $p_{n-1}(x_j) = y_j$ for $j = 1, \ldots, n$. We can write $p_n(x) = p_{n-1}(x) + q(x)$ and must find a polynomial $q(x)$ of degree $\leq n$ such that $q(x_1) = \cdots = q(x_n) = 0$. Therefore $q(x)$ must have the form

$$q(x) = a_n (x - x_1) \cdots (x - x_n)$$

(for each x_j we must have a factor $(x - x_j)$, the remaining factor must be a constant a_n since the degree of $q(x)$ is at most n). We therefore have to find c_n such that $p_n(x_{n+1}) = y_{n+1}$. This means that $q(x_{n+1}) = a_n (x_{n+1} - x_1) \cdots (x_{n+1} - x_n) = y_{n+1} - p_{n-1}(x_{n+1})$ which has the unique solution

$$a_n = \frac{y_{n+1} - p_{n-1}(x_{n+1})}{(x_{n+1} - x_1) \cdots (x_{n+1} - x_n)} \tag{2}$$

as $(x_n - x_1) \cdots (x_n - x_{n-1})$ is nonzero. □

Note that the proof does not just show existence, but actually gives an algorithm to construct the interpolating polynomial: We start with $p_0(x) = a_0$ where $a_0 = y_1$. Then determine a_1 from (2) and have $p_1(x) = a_0 + a_1(x - x_1)$. We continue in this way until we finally obtain

$$p_{n-1}(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2) + \cdots + a_{n-1}(x - x_1) \cdots (x - x_{n-1}). \tag{3}$$

This is the so-called **Newton form** of the interpolating polynomial. Once we know the coefficients $a_0, \ldots, a_{n-1}$ we can efficiently evaluate $p_{n-1}(x)$ using **nested multiplication**: E.g., for $n = 4$ we have

$$p_3(x) = ((a_3 \cdot (x - x_3) + a_2) \cdot (x - x_2) + a_1) \cdot (x - x_1) + a_0.$$

Nested multiplication algorithm for Newton form: Given interpolation nodes $x_1, \ldots, x_n$, Newton coefficients $a_0, \ldots, a_{n-1}$, evaluation point x, find $y = p_{n-1}(x)$.

$y := a_{n-1}$
For $j = n-1, n-2, \ldots, 1$:
  $y := y \cdot (x - x_j) + a_{j-1}$

Note that this algorithm takes $n - 1$ multiplications (and additions).

3.3 Divided differences and recursion formula

Multiplying out (3) gives

$$p_{n-1}(x) = a_{n-1}x^{n-1} + r(x)$$

where $r(x)$ is a polynomial of degree $\leq n-2$. We see that a_{n-1} is the **leading coefficient** (i.e., of the term x^{n-1}) of the interpolating polynomial p_{n-1}. For a given function f and nodes $x_1,\dots,x_{n-1}$ the interpolating polynomial p_{n-1} is uniquely determined, and in particular the leading coefficient a_{n-1}. We introduce the following **notation for the leading coefficient of an interpolating polynomial**:

$$f[x_1,\dots,x_n] = a_{n-1}$$

Examples: The notation $f[x_j]$ denotes the leading coefficient of the constant polynomial interpolating f in x_j, i.e.,

$$\boxed{f[x_j] = f(x_j)} \tag{4}$$

The notation $f[x_j,x_{j+1}]$ denotes the leading coefficient of the constant polynomial interpolating f in x_j, i.e.,

$$f[x_j,x_{j+1}] = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}.$$

In general the expression $f[x_1,\dots,x_m]$ is called a **divided difference**. Recall that the arguments $x_1,\dots,x_m$ must be different from each other. Note that the order of $x_1,\dots,x_n$ since there is only one interpolating polynomial, no matter in which order we specify the points.

Theorem 3.2. *There holds the* ***recursion formula***

$$\boxed{f[x_1,\dots,x_{m+1}] = \frac{f[x_2,\dots,x_{m+1}] - f[x_1,\dots,x_m]}{x_{m+1} - x_1}} \tag{5}$$

Proof. Let $p_{1,\dots,m}(x)$ denote the interpolating polynomial for the nodes $x_1,\dots,x_m$. Then we can construct the polynomial $p_{1,\dots,m+1}(x)$ for all nodes $x_1,\dots,x_{m+1}$ as

$$p_{1,\dots,m+1}(x) = p_{1,\dots,m}(x) + f[x_1,\dots,x_{m+1}]\cdot(x-x_1)\cdots(x-x_m).$$

Alternatively, we can start with the interpolating polynomial $p_{2,\dots,m+1}$ for the nodes $x_2,\dots,x_{m+1}$ and construct the polynomial $p_{1,\dots,m+1}(x)$ for all nodes $x_1,\dots,x_{m+1}$ as

$$p_{1,\dots,m+1}(x) = p_{2,\dots,m+1}(x) + f[x_1,\dots,x_{m+1}]\cdot(x-x_2)\cdots(x-x_{m+1}).$$

Taking the difference of the last two equations gives

$$0 = p_{1,\dots,m}(x) - p_{2,\dots,m+1}(x) + (x-x_2)\cdots(x-x_m)f[x_1,\dots,x_{m+1}]\cdot\underbrace{((x-x_1)-(x-x_{m+1}))}_{(x_{m+1}-x_1)}$$

$$\underbrace{p_{2,\dots,m+1}(x) - p_{1,\dots,m}(x)}_{f[x_2,\dots,x_{m+1}]x^m - f[x_1,\dots,x_m]x^m + O(x^{m-1})} = (x_{m+1}-x_1)\cdot f[x_1,\dots,x_{m+1}]\cdot x^m + O(x^{m-1})$$

$$(f[x_2,\dots,x_{m+1}] - f[x_1,\dots,x_m]) = (x_{m+1}-x_1)\cdot f[x_1,\dots,x_{m+1}]$$

where $O(x^{m-1})$ denotes polynomials of order $m-1$ or less. □

3.4 Divided difference algorithm

We now can compute any divided differences using (4) and (5). Given the nodes $x_1,\ldots x_n$ and function values $y_1,\ldots,y_n$ we can construct the **divided difference table** as follows: In the first column we write the nodes $x_1,\ldots,x_n$. In the next column we write the divided differences of 1 argument $f[x_1]=y_1,\ldots,f[x_n]=y_n$. In the next column we write the divided differences of 2 arguments $f[x_1,x_2],\ldots,f[x_{n-1},x_n]$ which we evaluate using (5). In the next column we write the divided differences of 3 arguments $f[x_1,x_2,x_3],\ldots,f[x_{n-2},x_{n-1},x_n]$ which we evaluate using (5). This continues until we write in the last column the single entry $f[x_1,\ldots,x_n]$.

$$\begin{array}{c|ccccc}
x_1 & f[x_1] & f[x_1,x_2] & f[x_1,x_2,x_3] & \cdots & f[x_1,\ldots,x_n] \\
\vdots & \vdots & \vdots & \vdots & \iddots & \\
\vdots & \vdots & \vdots & f[x_{n-2},x_{n-1},x_n] & & \\
\vdots & \vdots & f[x_{n-1},x_n] & & & \\
x_n & f[x_n] & & & &
\end{array}$$

Using the divided difference notation we can rewrite the Newton form (3) as

$$p_{n-1}(x)=f[x_1]+f[x_1,x_2](x-x_1)+\cdots+f[x_1,\ldots,x_n](x-x_1)\cdots(x-x_{n-1}).$$

Note that this formula uses the **top entries of each column of the divided difference table**.

However, we can also consider the nodes in the reverse order $x_n,x_{n-1},\ldots,x_1$ and obtain the alternative Newton form

$$p_{n-1}(x)=f[x_n]+f[x_{n-1},x_n](x-x_n)+\cdots+f[x_1,\ldots,x_n](x-x_n)(x-x_{n-1})\cdots(x-x_2)$$

for the same polynomial $p_{n-1}(x)$. Note that this formula uses the **bottom entries of each column of the divided difference table**.

Let us use this second formula. We can implement this just storing n numbers $d_1,\ldots,d_n$. We can first compute the first column $d_1,\ldots,d_n$, then we compute the second column overwriting $d_1,\ldots,d_{n-1},\ldots$, the last column overwriting d_1:

$$\begin{array}{ccccc}
d_1 := f[x_1] & d_1 := f[x_1,x_2] & d_1 := f[x_1,x_2,x_3] & \cdots & d_1 := f[x_1,\ldots,x_n] \\
\vdots & \vdots & \vdots & \iddots & \\
\vdots & \vdots & d_{n-2} := f[x_{n-2},x_{n-1},x_n] & & \\
\vdots & d_{n-1} := f[x_{n-1},x_n] & & & \\
d_n := f[x_n] & & & &
\end{array}$$

In the end we have $d_n=f[x_n]$, $d_{n-1}=f[x_{n-1},x_n],\ldots,$ $d_1=f[x_1,\ldots,x_n]$ so that

$$p_{n-1}(x)=d_n+d_{n-1}(x-x_n)+d_{n-2}(x-x_n)(x-x_{n-1})+\cdots+d_1(x-x_n)\cdots(x-x_2)$$

Divided difference algorithm, Part 1: Given $x_1,\ldots,x_n$, $y_1,\ldots,y_n$ find the Newton coefficients $d_1,\ldots,d_n$
For $i=1,\ldots,n$ do:
  $d_i := y_i$
For $k=1,\ldots,n-1$ do:
  For $i=1,\ldots,n-k$do:
    $d_i = \dfrac{d_{i+1}-d_i}{x_{i+k}-x_i}$

Divided difference algorithm, Part 2: Given $x_1,\ldots,x_n$, $d_1,\ldots,d_n$ and an evaluation point x find $y=p_{n-1}(x)$
$y := d_1$
For $i=2,\ldots,n$:
  $y := y\cdot(x-x_i)+d_i$

This gives the following Matlab code:

```
function d = divdiff(x,y)
% compute Newton form coefficients of interpolating polynomial
n = length(x);
d = y;
for k=1:n-1
  for i=1:n-k
    d(i) = (d(i+1)-d(i))/(x(i+k)-x(i));
  end
end
```

```
function yt = evnewt(d,x,xt)
% evaluate Newton form of interpolating polynomial at points xt
yt = d(1)*ones(size(xt));
for i=2:length(d)
  yt = yt.*(xt-x(i)) + d(i);
end
```

Example: We are given the data points

x_j	0	1	2	4
y_j	1	2	3	1

. Find the interpolating polynomial in Newton form.

We enter the x_j values in the first column and the y_j values in the second column:

x_j	$f[x_j]$	$f[x_j,x_{j+1}]$	$f[x_j,x_{j+1},x_{j+2}]$	$f[x_1,x_2,x_3,x_4]$
0	1	1	0	$-\frac{1}{6}$
1	2	1	$-\frac{2}{3}$	
2	3	-1		
4	1			

We then obtain the remaining columns by using the recursion formula.

For the nodes in order x_1,x_2,x_3,x_4 we obtain the Newton form

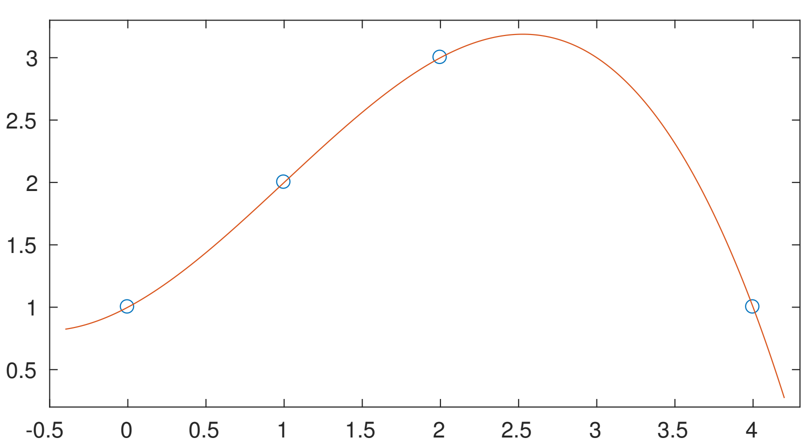
$$\begin{aligned} p(x) &= f[x_1]+f[x_1,x_2](x-x_1)+f[x_1,x_2,x_3](x-x_1)(x-x_2)+f[x_1,x_2,x_3,x_4](x-x_1)(x-x_2)(x-x_3) \\ &= 1+1\cdot(x-0)+0\cdot(x-0)(x-1)+(-\tfrac{1}{6})(x-0)(x-1)(x-2) \end{aligned}$$

For the nodes in order x_4,x_3,x_2,x_1 we obtain the Newton form

$$\begin{aligned} p(x) &= f[x_4]+f[x_3,x_4](x-x_4)+f[x_2,x_3,x_4](x-x_4)(x-x_3)+f[x_1,x_2,x_3,x_4](x-x_4)(x-x_3)(x-x_2) \\ &= 1+(-1)(x-4)+(-\tfrac{2}{3})(x-4)(x-2)+(-\tfrac{1}{6})(x-4)(x-2)(x-1) \end{aligned}$$

In Matlab we can plot the given points and the interpolating polynomial as follows:

```
x = [0,1,2,4]; y = [1,2,3,1]; % given x and y values
d = divdiff(x,y)              % find coefficients of Newton form
xt = -.4:.01:4.2;             % x-values for plotting
yt = evnewt(d,x,xt);          % evaluate Newton form at points xt
plot(x,y,'o',xt,yt)           % plot given pts and interpolating polynomial
```



3.5 Error formula for $f(x) - p(x)$

A divided difference $f[x_j, x_{j+1}]$ of two arguments satisfies

$$f[x_j, x_{j+1}] = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j} = f'(s)$$

for some $s \in (x_j, x_{j+1})$ by the mean value theorem. For general divided differences we have a similar result:

Theorem 3.3. *Assume that the derivatives $f, f', \ldots, f^{(n-1)}$ exist and are continuous. Let $x_1, \ldots, x_n$be different from each other. Then there exists $s \in (\min\{x_1, \ldots, x_n\}, \max\{x_1, \ldots, x_n\})$ such that*

$$f[x_1, \ldots, x_n] = \frac{f^{(n-1)}(s)}{(n-1)!}. \tag{6}$$

Proof. Consider the interpolating polynomial $p(x)$ and the interpolation error $e(x) = f(x) - p(x)$. Then the function $e(x)$ is zero for $x_1, \ldots, x_n$, hence it has at least n different zeros.

Since $e(x_1) = 0$ and $e(x_2) = 0$ there exists by the mean value theorem a point $x_1' \in (x_1, x_2)$ with $e'(x_1') = 0$. Hence the function $e'(x)$ has at least $n-1$ different zeros. Similarly, the function $e''(x)$ has at at least $n-2$ different zeros,..., the function $e^{(n-1)}$ has at least one zero s. Hence we have

$$0 = e^{(n-1)}(s) = f^{(n-1)}(s) - p^{(n-1)}(s).$$

Since $p(x) = f[x_1, \ldots, x_n]x^{n-1} + O(x^{n-2})$ we have $p^{(n-1)}(x) = f[x_1, \ldots, x_n](n-1)!$. □

Let $x_1, \ldots, x_n$ be different from each other and let $p_{n-1}(x)$ be the interpolating polynomial for the function $f(x)$. Let $\tilde{x}$ be different from $x_1, \ldots, x_n$. We want to find a formula for the interpolation error $f(\tilde{x}) - p_{n-1}(\tilde{x})$: We first construct an interpolating polynomial $p_n(x)$ which interpolates in the points $x_1, \ldots, x_n$ and $\tilde{x}$. We must have

$$p_n(x) = p_{n-1}(x) + f[x_1, \ldots, x_n, \tilde{x}](x - x_1) \cdots (x - x_n)$$

and using $f(\tilde{x}) = p_n(\tilde{x})$ we obtain

$$f(\tilde{x}) - p_{n-1}(\tilde{x}) = f[x_1, \ldots, x_n, \tilde{x}](\tilde{x} - x_1) \cdots (\tilde{x} - x_n).$$

We can now express the divided difference using (6) and obtain

Theorem 3.4. *Assume that the derivatives $f, f', \ldots, f^{(n)}$ exist and are continuous. Let $x_1, \ldots, x_n$be different from each other and let p_{n-1} denote the interpolating polynomial. Then there exists an intermediate point $s \in (\min\{x_1, \ldots, x_n, \tilde{x}\}, \max\{x_1, \ldots, x_n, \tilde{x}\})$ such that*

$$f(\tilde{x}) - p_{n-1}(\tilde{x}) = \frac{f^{(n)}(s)}{n!} \cdot (\tilde{x} - x_1) \cdots (\tilde{x} - x_n).$$

The function $\omega(x) := (x - x_1) \cdots (x - x_n)$ is called the **node polynomial**.

In practice we don't know where the intermediate point s is located. If we know that $x_1, \ldots, x_n$ and $\tilde{x}$ are in an interval $[a, b]$ we have the (possibly very pessimistic) upper bound

$$|f(\tilde{x}) - p(\tilde{x})| \leq \frac{1}{n!} \left(\max_{s \in [a,b]} \left| f^{(n)}(s) \right| \right) \cdot |\omega(\tilde{x})|$$

- The first term depends only on the function f and not on the nodes. This term becomes zero if $f^{(n)} = 0$ which happens if and only if f is a polynomial of degree $\leq n-1$. In this case we must have $p_{n-1}(x) = f(x)$ since the interpolating polynomial is unique.
- The second term $|\omega(\tilde{x})|$ depends only on $\tilde{x}$ and the nodes $x_1, \ldots, x_n$ (and not on f). This term becomes equal to zero at the nodes, and it is small if $\tilde{x}$ is close to one of the nodes.

3.6 Interpolation with multiple nodes

So far we assumed that the nodes $x_1, \dots, x_n$ are different from each other. What happens if we move two nodes closer and closer together?

Example 1: Consider three nodes $x_1 < x_2 < x_3$. In this case we have the divided difference table

$$\begin{array}{c|lll}
x_1 & f(x_1) & f[x_1,x_2] = \frac{f(x_2)-f(x_1)}{x_2-x_1} & f[x_1,x_2,x_3] = \frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1} \\
x_2 & f(x_2) & f[x_2,x_3] = \frac{f(x_3)-f(x_2)}{x_3-x_2} & \\
x_3 & f(x_3) & &
\end{array}$$

and the interpolating polynomial $p(x) = f[x_1] + f[x_1,x_2](x-x_1) + f[x_1,x_2,x_3](x-x_1)(x-x_2)$.

Now we move the node x_2 towards x_1 and want to know what happens in the limit. Assume that the function f is differentiable, then we get for $f[x_1,x_2]$

$$\lim_{x_2 \to x_1} \frac{f(x_2) - f(x_1)}{x_2 - x_1} = f'(x_1)$$

Hence we *define* $f[x_1,x_1] = f'(x_1)$. The divided difference table becomes

$$\begin{array}{c|lll}
x_1 & f(x_1) & f[x_1,x_1] = f'(x_1) & f[x_1,x_1,x_3] = \frac{f[x_1,x_3]-f[x_1,x_1]}{x_3-x_1} \\
x_1 & f(x_1) & f[x_1,x_3] = \frac{f(x_3)-f(x_1)}{x_3-x_1} & \\
x_3 & f(x_3) & &
\end{array}$$

and the interpolating polynomial is $p(x) = f[x_1] + f[x_1,x_1](x-x_1) + f[x_1,x_1,x_3](x-x_1)(x-x_1)$. This function still satisfies $p(x_1) = f(x_1)$ and $p(x_2) = f(x_2)$. Additionally we have $p'(x_1) = f[x_1,x_1] = f'(x_1)$. Therefore $p(x)$ solves the following problem:

Given $f(x_1), f'(x_1), f(x_3)$ find an interpolating polynomial

Example 2: Consider nodes $x_1 = x_2 = x_3 < x_4 = x_5$.

$$\begin{array}{c|lllll}
x_1 & f(x_1) & f[x_1,x_1] = f'(x_1) & f[x_1,x_1,x_1] = \frac{1}{2}f''(x_1) & \cdots & f[x_1,x_1,x_1,x_4,x_4] \\
x_1 & f(x_1) & f[x_1,x_1] = f'(x_1) & f[x_1,x_1,x_4] = \frac{f[x_1,x_4]-f[x_1,x_1]}{x_4-x_1} & \iddots & \\
x_1 & f(x_1) & f[x_1,x_4] = \frac{f(x_4)-f(x_1)}{x_4-x_1} & f[x_1,x_4,x_4] = \frac{f[x_4,x_4]-f[x_1,x_4]}{x_4-x_1} & & \\
x_4 & f(x_4) & f[x_4,x_4] = f'(x_4) & & & \\
x_4 & f(x_4) & & & &
\end{array}$$

Summary:

- For the nodes $x_1 \le x_2 \le \cdots \le x_n$ we now allow multiple nodes. For a node x_j of multiplicity m we are given $f(x_j), f'(x_j), \dots, f^{(m-1)}(x_j)$.
- We want to find an interpolating polynomial $p(x)$ of degree$\le n-1$ which satisfies the n conditions for the function values and derivatives. This interpolation problem has a unique solution $p(x)$.
- We define divided differences with m identical nodes

$$f[x_j, \cdots, x_j] := \frac{f^{(m-1)}(x_j)}{(m-1)!}$$

- Using this definition, we can fill the whole divided difference table and then obtain

$$p(x) = f[x_1] + f[x_1,x_2](x-x_1) + \cdots + f[x_1,\dots,x_n](x-x_1)\cdots(x-x_{n-1})$$

- The error formula also holds for multiple nodes:

$$f(x) - p(x) = \frac{f^{(n)}(t)}{n!}(x-x_1)\cdots(x-x_n)$$

  where t is between the points $x, x_1, \dots, x_n$