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
  - evaluate $f(x_1), \ldots, f(x_n)$ by some (possibly costly) approximation method (e.g. Taylor series or other series)
  - find the interpolating polynomial $p(x)$ and store its coefficients

  Now we have a simple way to compute an approximation for $f(x)$:
  - given $x \in [a,b]$ use the stored coefficients to evaluate the polynomial $p(x)$

  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*

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

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 $x_1,\ldots,x_{n-1}$ 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**:

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

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 $f[x_1,\ldots,x_m]$ is called a **divided difference**. Recall that the arguments $x_1,\ldots,x_m$ must be different from each other. The order of the arguments $x_1,\ldots,x_n$ does not matter since there is only one interpolating polynomial, no matter in which order we specify the points.

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

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

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

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

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

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

Taking the difference of the last two equations gives

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

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

where $O(x^{m-2})$ denotes polynomials of order $m-2$ or less. □

3.4 Divided difference algorithm

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

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

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

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

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 $x_n,x_{n-1},\dots,x_1$ and obtain the alternative Newton form

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

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 $d_1,\dots,d_n$. We can first compute the first column $d_1,\dots,d_n$, then we compute the second column overwriting $d_1,\dots,d_{n-1},\dots$, the last column overwriting d_1:

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

In the end we have $d_n = f[x_n]$, $d_{n-1} = f[x_{n-1},x_n],\dots,$ $d_1 = f[x_1,\dots,x_n]$ 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 $x_1,\dots,x_n$, $y_1,\dots,y_n$ find the Newton coefficients $d_1,\dots,d_n$
For $i=1,\dots,n$ do:
  $d_i := y_i$
For $k=1,\dots,n-1$ do:
  For $i=1,\dots,n-k$ do:
    $$d_i = \frac{d_{i+1}-d_i}{x_{i+k}-x_i}$$

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

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 $\begin{array}{c|cccc} x_j & 0 & 1 & 2 & 4 \\ \hline y_j & 1 & 2 & 3 & 1 \end{array}$. 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

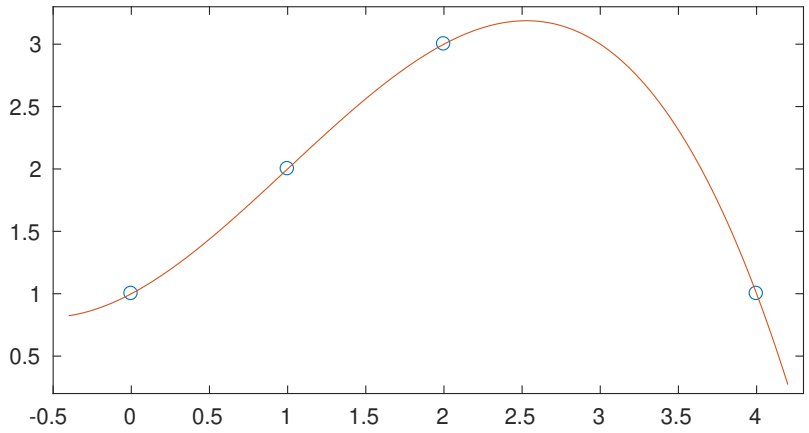
$$\begin{aligned} p(x) &= f[x_1]+f[x_1,x_2](x-x_1)+f[x_1,x_2,x_3](x-x_1)(x-x_2)+f[x_1,x_2,x_3,x_4](x-x_1)(x-x_2)(x-x_3) \\ &= 1+1\cdot(x-0)+0\cdot(x-0)(x-1)+(-\tfrac{1}{6})(x-0)(x-1)(x-2) \end{aligned}$$

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*

$$f(\tilde{x}) - p_{n-1}(\tilde{x}) = \frac{f^{(n)}(s)}{n!} \cdot (\tilde{x} - x_1) \cdots (\tilde{x} - x_n). \tag{7}$$

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 x are in an interval $[a, b]$ we have the upper bound (which may be way too large):

$$|f(x) - p(x)| \leq \frac{1}{n!} \left(\max_{s \in [a,b]} \left| f^{(n)}(s) \right| \right) \cdot |\omega(x)| \tag{8}$$

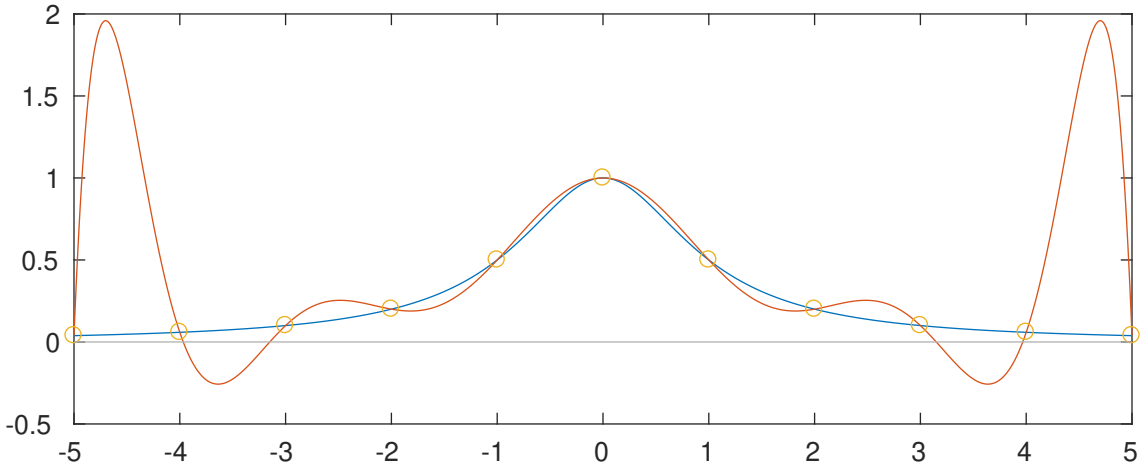
- The first term $\frac{1}{n!} \left(\max_{s \in [a,b]} \left| f^{(n)}(s) \right| \right)$ depends only on the function f and not on the nodes. This term becomes zero if $f^{(n)}(x) = 0$ for all x, hence f is a polynomial of degree $\leq n-1$. In this case we must have $p_{n-1}(x) = f(x)$ since the interpolating polynomial is unique.
- The second term $|\omega(x)|$ depends only on 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 x is close to one of the nodes.

3.6 Equidistant nodes and Chebyshev nodes

We want to approximate a function $f(x)$ for $x \in [a,b]$: We choose nodes $x_1,\dots,x_n \in [a,b]$ and construct the interpolating polynomial $p_{n-1}(x)$. How should we choose the nodes $x_1,\dots,x_n$ in order to have a small interpolation error?

We could try **equidistant nodes:** We divide the interval $[a,b]$ into $n-1$ subintervals of length $h = \dfrac{b-a}{n-1}$ and use the nodes $x_j = a+(j-1)h$ for $j = 1,\dots,n$.

Example: We consider the function $f(x) = \dfrac{1}{1+x^2}$ on the interval $[-5,5]$. Note that this function has derivatives of any order and is analytic: for any $x_0 \in \mathbb{R}$ the Taylor series about x_0 converges to $f(x)$ in a neighborhood of x_0. We use $n = 11$ equidistant nodes $x_1 = -5, x_2 = -4, \dots, x_{11} = 5$ and obtain the following interpolating polynomial



Here we divided $[-5,5]$ into 10 subintervals, and the maximal error is $E_{10} := \max_{[-5,5]} |f(x) - p_{10}(x)| \approx 1.96$, due to the large oscillations near the endpoints. We now bisect the 10 subintervals, adding a new node at each midpoint. Now we get the maximal error $E_{20} := \max_{[-5,5]} |f(x) - p_{20}(x)| \approx 59.8$. If we bisect again and again we get the maximal errors

$$E_{10} \approx 1.96, \qquad E_{20} \approx 59.8, \qquad E_{40} \approx 1.0 \cdot 10^5, \qquad E_{80} \approx 5.5 \cdot 10^{11}, \qquad E_{160} \approx 2.5 \cdot 10^{25}, \qquad E_{320} \approx 8.1 \cdot 10^{52}$$

The size of the oscillations near the endpoints $-5, 5$ gets worse and worse, growing exponentially with n.

What is going on? The error formula (7) contains the node polynomial $\omega(x) = (x-x_1)\cdots(x-x_n)$.

Claim: For equidistant nodes the node polynomial $\omega(x)$ has huge oscillations near the endpoints and very small oscillations near the center.
Let us consider for example the 10 equidistant nodes $1,2,3,4,5,6,7,8,9,10$. Then the maximal value in the center interval occurs at the midpoint $t_c = 5.5$ where we have

$$|\omega(t_c)| = (0.5 \times 1.5 \times 2.5 \times 3.5 \times 4.5)^2$$

In the midpoint $t_1 = 1.5$ of the first interval we have

$$|\omega(t_1)| = 0.5 \times 0.5 \times 1.5 \times 2.5 \times 3.5 \times 4.5 \times 5.5 \times 6.5 \times 7.5 \times 8.5$$

Hence we have for $n = 10$

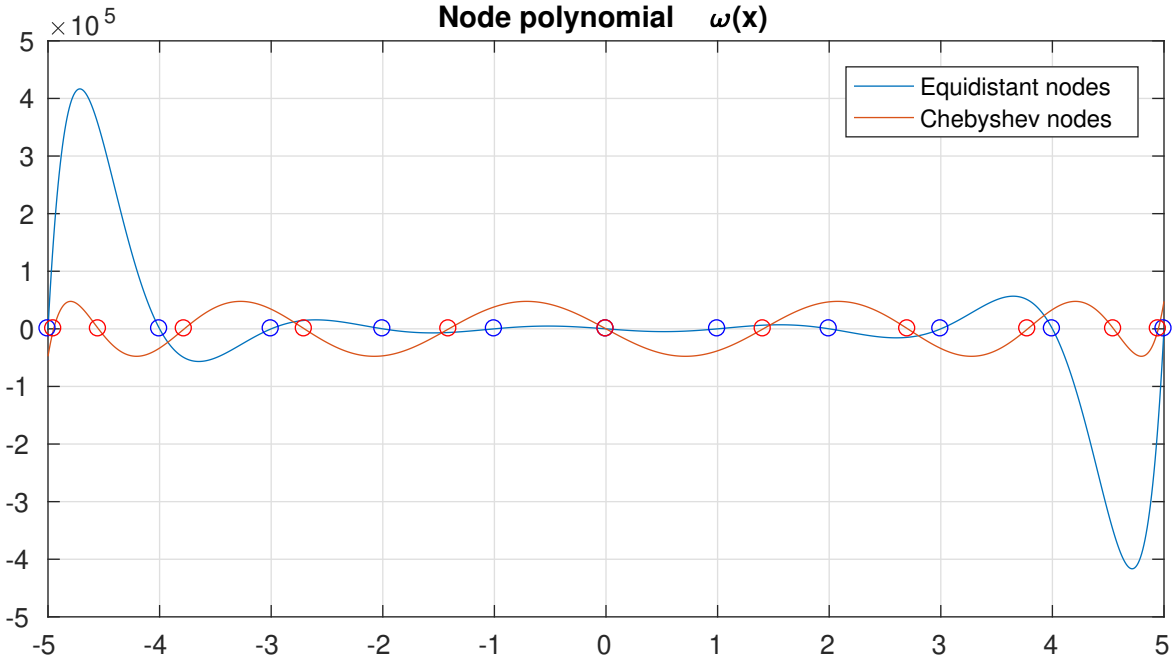
$$\frac{|\omega(t_1)|}{|\omega(t_c)|} = \frac{8.5}{4.5} \times \frac{7.5}{3.5} \times \frac{6.5}{2.5} \times \frac{5.5}{1.5} \geq 2^4$$

(the first factor is slightly smaller than 2, but the product of the first two factors is$> 2^2$). In the same way we obtain for any even n that $\frac{|\omega(t_1)|}{|\omega(t_c)|} \geq 2^{\frac{n}{2}-1}$.

Therefore the maximum of $|\omega(x)|$ in the first interval is at least by a factor of $2^{\frac{n}{2}-1}$ larger than the maximum in the center interval.

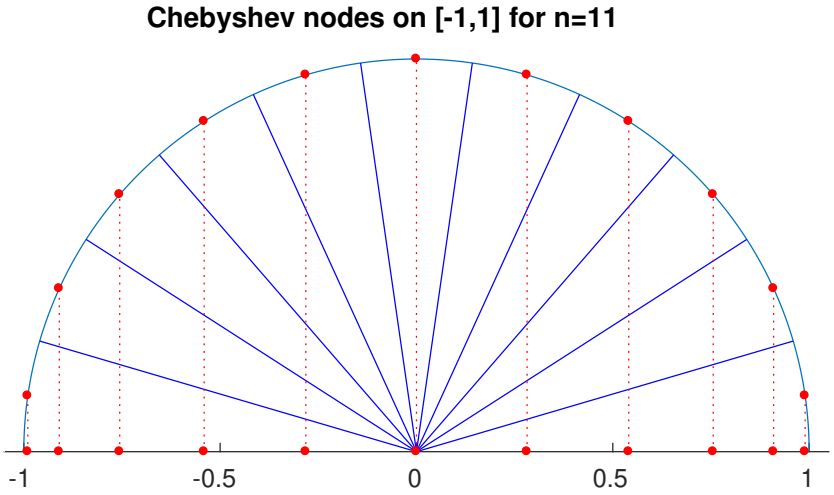
For equidistant nodes we get huge values for $|\omega(x)|$ near the endpoints, and very small values for $|\omega(x)|$ near the center of the interval $[a,b]$. **We want to move the nodes so that we get smaller values near the endpoints, and larger values near the center.** We can do this by moving the nodes closer together near the endpoints, and moving farther apart near the center.

It turns out that one can find an optimal choice of nodes such that $\omega_{\max} = \max_{x\in[a,b]} |\omega(x)|$ is as small as possible. For this choice of nodes the maxima of $|\omega(x)|$ have the same value in all of the intervals between the nodes.



The optimal choice of nodes are the so-called **Chebyshev nodes** $x_1,\ldots,x_n$ given by

$$x_j = \frac{a+b}{2} + \frac{b-a}{2}\cos\left((j-\tfrac{1}{2})\frac{\pi}{n}\right), \qquad j=1,\ldots,n$$



Note that the Chebyshev nodes are closer together near the endpoints, and farther apart near the center.

We claim:

- for the Chebyshev nodes the node polynomial the local maxima of $|\omega(x)|$ have the same size $\omega_{\max}$ on each subinterval (x_j, x_{j+1})
- the maximum of the node polynomial is $\omega_{\max} = \max\limits_{x\in[a,b]} |\omega(x)| = 2\left(\dfrac{b-a}{4}\right)^n$
- **this value of $\omega_{\max}$ is optimal**: it is not possible to find nodes $x_1,\ldots,x_n$ with a smaller value of $\omega_{\max}$.

3.7 Chebyshev nodes: Theoretical background (you can skip this section)

Chebyshev polynomials $T_k(x)$

We consider the mapping $x = \cos t$ for $t \in [0,\pi]$. This gives a one-to-one mapping from $[0,\pi]$ to $[-1,1]$. We denote the inverse mapping $[-1,1] \to [0,\pi]$ by $t = \cos^{-1}(x)$.

We define for $n = 0,1,2,\ldots$ the functions

$$\boxed{T_n(x) := \cos\left(n \cdot \cos^{-1}(x)\right)} \tag{9}$$

We have for $n = 0$ and $n = 1$

$$T_0(x) = \cos(0) = 1, \qquad T_1(x) = \cos\left(\cos^{-1}(x)\right) = x \tag{10}$$

Let $t = \cos^{-1}(x)$. Then we can use the formula $\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$ for $T_{n-1}(x)$, $T_{n+1}(x)$:

$$\begin{aligned} T_{n-1}(x) &= \cos\left((n-1)t\right) = \cos(nt)\cos(t) + \sin(nt)\sin(t) \\ T_{n+1}(x) &= \cos\left((n+1)t\right) = \cos(nt)\cos(t) - \sin(nt)\sin(t) \\ T_{n-1}(x) + T_{n+1}(x) &= 2\underbrace{\cos(nt)}_{T_n(x)}\underbrace{\cos(t)}_{x} \end{aligned}$$

yielding the **recursion formula**

$$\boxed{T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)} \tag{11}$$

Using (10), (11) we can find $T_n(x)$ for any $n = 0,1,2,3,\ldots$:

$$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_2(x) = 2x^2 - 1, \qquad T_3(x) = 4x^3 - 3x, \qquad T_4(x) = 8x^4 - 8x^2 + 1, \qquad T_5(x) = 16x^5 - 20x^3 + 5x$$

$T_n(x)$ is called **Chebyshev polynomial** of degree n. We have

$$T_0(x) = 1, \qquad \text{for } n \geq 1: \quad T_n(x) = 2^{n-1}x^n + \text{lower order terms} \tag{12}$$

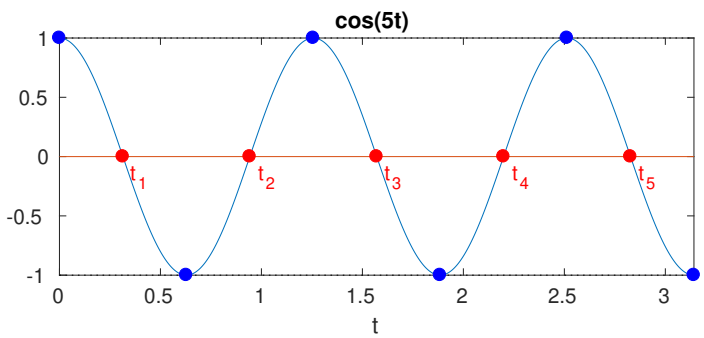
Note that the function $T_n(x)$ for $x \in [-1,1]$ is related to the function $\cos(nt)$ for $t \in [0,\pi]$ by the change of variable $x = \cos t$.

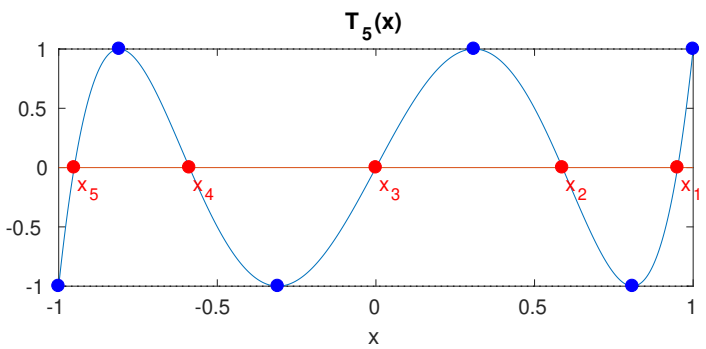
The function $f(t) = \cos(nt)$ for $t \in [0,\pi]$ satisfies $|f(t)| \leq 1$ and has

- the n **zeros** $t_k = (k - \frac{1}{2})\frac{\pi}{n}$, $k = 1,\ldots n$ with $f(t_k) = 0$
- the $n+1$ **extrema** $\tilde{t}_k = k\frac{\pi}{n}$, $k = 0,\ldots,n$ with $f(\tilde{t}_k) = (-1)^k$

Therefore the function $T_n(x)$ for $x \in [-1,1]$ satisfies $|T_n(x)| \leq 1$ and has

- the n **zeros** $x_k = \cos(t_k) = \cos\left((k - \frac{1}{2})\frac{\pi}{n}\right)$, $k = 1,\ldots n$ with $T_n(x_k) = 0$
- the $n+1$ **extrema** $\tilde{x}_k = \cos(\tilde{t}_k) = \cos\left(k\frac{\pi}{n}\right)$ $k = 0,\ldots,n$ with $T_n(\tilde{x}_k) = (-1)^k$





Note that $(x - x_1)\cdots(x - x_n) = x^n + \text{lower order terms}$. Because of (12) we have for $n \geq 1$

$$(x - x_1)\cdots(x - x_n) = 2^{1-n}T_n(x) \tag{13}$$

Chebyshev nodes for interpolation on $[-1,1]$

We want to approximate a function $f(x)$ for $x \in [-1,1]$: We choose nodes $x_1,\dots,x_n \in [-1,1]$ and construct the interpolating polynomial $p_{n-1}(x)$. How should we choose the nodes? The maximal error on the interval

$$E_{\max} := \max_{x\in[-1,1]} |f(x) - p_{n-1}(x)|$$

should be small. From (8) we obtain the upper bound

$$E_{\max} \le \frac{1}{n!}\left(\max_{s\in[a,b]}\left|f^{(n)}(s)\right|\right)\cdot \underbrace{\left(\max_{x\in[-1,1]}|\omega(x)|\right)}_{\omega_{\max}}$$

where $\omega(x) = (x-x_1)\cdots(x-x_n)$ is the so-called node polynomial. In order to obtain a small bound for $E_{\max}$ we therefore want to

$$\boxed{\text{pick nodes } x_1,\dots,x_n \text{ such that } \omega_{\max} \text{ is small}}$$

We now pick the **Chebyshev nodes** $\boxed{x_k = \cos\left((k-\frac{1}{2})\frac{\pi}{n}\right)}$ for $k = 1,\dots,n$. Because of (13) and $|T_n(x)| \le 1$ for $x \in [-1,1]$ we obtain

$$\omega_{\max} = \max_{x\in[-1,1]} |\omega(x)| = 2^{1-n}$$

Can we find nodes $\hat{x}_1,\dots,\hat{x}_n$ with $\hat{\omega}_{\max} = \max_{[-1,1]}|(x-\hat{x}_1)\cdots(x-\hat{x}_n)| < 2^{1-n}$?
Assume that $\hat{\omega}_{\max} < \omega_{\max} = 2^{1-n}$ and consider the difference polynomial $q(x) = \omega(x) - \hat{\omega}(x)$. Since $\omega(x) = x^n + \text{l.o.t.}$ and $\hat{\omega}(x) = x^n + \text{l.o.t.}$ the polynomial $q(x)$ is of degree$\le n-1$. Recall that $\omega(x) = 2^{1-n}T_n(x)$ has the extremal values $\omega(\tilde{x}_k) = (-1)^k 2^{1-n}$, $k = 0,\dots,n$ where $\tilde{x}_k = \cos\left(k\frac{\pi}{n}\right)$. Consider the interval $[\tilde{x}_1,\tilde{x}_0]$: at the endpoints $\tilde{x}_0,\tilde{x}_1$ we have

$$q(\tilde{x}_0) = \underbrace{\omega(\tilde{x}_0)}_{2^{1-n}} - \underbrace{\hat{\omega}(\tilde{x}_0)}_{<2^{1-n}} > 0, \qquad q(\tilde{x}_1) = \underbrace{\omega(\tilde{x}_1)}_{-2^{1-n}} \underbrace{-\hat{\omega}(\tilde{x}_0)}_{<2^{1-n}} < 0$$

and by the intermediate value theorem the function $q(x)$ must have a zero in the interval $(\tilde{x}_1,\tilde{x}_0)$. By the same argument the function $q(x)$ has a zero in the intervals $(\tilde{x}_k,\tilde{x}_{k-1})$ for $k = 1,\dots,n$. Therefore the polynomial $q(x)$ has at least n distinct zeros. But $q(x)$ is a polynomial of degree$\le n-1$, and therefore we must have $q(x) = 0$ for all x. But this means that $\omega(x) = \hat{\omega}(x)$ which is a contradiction to our assumption $\hat{\omega}_{\max} < \omega_{\max}$. We have proved the following result:

Theorem 3.5. *The nodes $x_1,\dots,x_n \in [-1,1]$ which give the smallest value*

$$\omega_{\max} = \max_{x\in[-1,1]} |(x-x_1)\cdots(x-x_n)|$$

are the Chebyshev nodes given by $x_k = \cos\left((k-\frac{1}{2})\frac{\pi}{n}\right)$, yielding the minmal value $\omega_{\max} = 2^{1-n}$.

If we want to approximate a function $f(x)$ **on an interval** $x \in [a,b]$ instead of $[-1,1]$ we use the mapping $x \mapsto \frac{a+b}{2} + x\frac{b-a}{2}$. This maps $x \in [-1,1]$ to the interval $[a,b]$. We obtain for the optimal choice of nodes on $[a,b]$:
The **Chebyshev nodes for the interval** $[a,b]$ are given by $\boxed{x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\left((k-\frac{1}{2})\frac{\pi}{n}\right)}$ and we have that

$$\omega_{\max} = \max_{x\in[a,b]} |(x-x_1)\cdots(x-x_n)| = \left(\frac{b-a}{2}\right)^n 2^{1-n} = 2\left(\frac{b-a}{4}\right)^n$$

3.8 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

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

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

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

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 $p'(x_1)=f[x_1,x_1]=f'(x_1)$.

Note that the blue values in the divided difference table are given, and the black values are obtained by the recursion formula.

Therefore $p(x)$ solves an **interpolation problem where function values and some derivative values are given:**

Find an interpolating polynomial $p(x)$ satisfying

$$\boxed{p(x_1)=f(x_1),\quad p'(x_1)=f'(x_1),\qquad p(x_3)=f(x_3)}$$

where x_1,x_3 and $f(x_1),f'(x_1),f(x_3)$ are given.

Example 2: Find an interpolating polynomial $p(x)$ satisfying the $n=5$ conditions

$$\boxed{p(1)=3,\quad p'(1)=4,\quad p''(1)=5\qquad p(2)=6,\quad p'(2)=7}$$

Now 1 **is a triple node** and 2 **is a double node**: we have $n=5$ nodes $\boxed{1,1,1,2,2}$

$$\begin{array}{l|llll} 1 & f[1]=3 & f[1,1]=4 & f[1,1,1]=\frac{5}{2} & f[1,1,1,2]=\frac{f[1,1,2]-f[1,1,1]}{2-1} \qquad f[1,1,1,2,2]=\frac{f[1,1,2,2]-f[1,1,1,2]}{2-1} \\ 1 & f[1]=3 & f[1,1]=4 & f[1,1,2]=\frac{f[1,2]-f[1,1]}{2-1} & f[1,1,2,2]=\frac{f[1,2,2]-f[1,1,2]}{2-1} \\ 1 & f[1]=3 & f[1,2]=\frac{f[2]-f[1]}{2-1} & f[1,2,2]=\frac{f[2,2]-f[1,2]}{2-1} & \\ 2 & f[2]=6 & f[2,2]=7 & & \\ 2 & f[2]=6 & & & \end{array}$$

Note that the blue values in the divided difference table are given, and the black values are obtained by the recursion formula. We now obtain the **interpolating polynomial in Newton form:** if we use the nodes in the order $1,1,1,2,2$ we use the **top entries of each column** and get

$$p(x)=f[1]+f[1,1](x-1)+f[1,1,1](x-1)(x-1)+f[1,1,1,2](x-1)(x-1)(x-1)+f[1,1,1,2,2](x-1)(x-1)(x-1)(x-2)$$

We can also use the nodes in the **reverse order** and use the **bottom entries of each column**. This gives the Newton form

$$p(x)=f[2]+f[2,2](x-2)+f[1,2,2](x-2)(x-2)+f[1,1,2,2](x-1)(x-2)(x-2)+f[1,1,1,2,2](x-1)(x-1)(x-2)(x-2)$$

Summary: Interpolation with multiple nodes

- At a node X the values $f(X), f'(X), \ldots, f^{(m-1)}(X)$ are given. We say that node X has **multiplicity** m. This means that we use $\underbrace{X, \ldots, X}_{m \text{ times}}$ in the list of nodes $x_1, \ldots, x_n$.
- Let n be the sum of the multiplicities of all the nodes. We want to find an interpolating polynomial $p(x)$ of degree$\leq n-1$ which satisfies 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** as follows:

$$\boxed{f[\underbrace{X, \ldots, X}_{m \text{ times}}] := \frac{f^{(m-1)}(X)}{(m-1)!}}$$

- **Algorithm:** First fill in the divided differences with identical nodes using the given data. Then fill in the remaining entries of the divided difference table using the recursion formula (5). Now the interpolating polynomial is given by

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

  We can also use the nodes in reverse order. This gives the Newton form

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

- The error formula also holds for multiple nodes:

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

  where $\boxed{\min\{x_1, \ldots, x_n, x\} \leq t \leq \max\{x_1, \ldots, x_n, x\}}$