Polynomial Interpolation, Chapter 4.1

(We do not cover the topic *divided differences* from page 160 to 167 and the last part from page 170 to 173)

Why and when do we need to find a function like a polynomial or other type functions to represent the data?

Polynomial Interpolation, Chapter 4.1

(We do not cover the topic *divided differences* from page 160 to 167 and the last part from page 170 to 173)

Why and when do we need to find a function like a polynomial or other type functions to represent the data?

- K1 We only have a finite number of data. If we can find a simple formula to reproduce the given points exactly, then we can estimate the value y for any x.
- K2 If the data are corrupted by errors, we look for a formula that represents the data and filters out the errors.
- K3 A function p is given, but expensive to evaluate. Then, we can find another simpler function to approximate p, which is easier to be calculated or integrated or differentiated.

Polynomial Interpolation

Consider a table of values:

x	x_0	x_1	$\ldots$	x_n
y	y_0	y_1	$\ldots$	y_n

We assume that the x_i's form a set of $n+1$ distinct points, called as **nodes**, i.e., $x_i \neq x_j$ if $i \neq j$.

A function p **interpolates** data $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, if

$$p(x_i) = y_i \quad \text{for } i = 0, \ldots, n.$$

Polynomial Interpolation

Consider a table of values:

x	x_0	x_1	$\dots$	x_n
y	y_0	y_1	$\dots$	y_n

We assume that the x_i's form a set of $n+1$ distinct points, called as **nodes**, i.e., $x_i \neq x_j$ if $i \neq j$.

A function p **interpolates** data $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, if

$$p(x_i) = y_i \quad \text{for } i = 0, \dots, n.$$

- Which degree polynomial should we use to interpolate the data?
- Is the interpolating polynomial unique?
- How can we calculate the interpolating polynomial?
- How can we calculate the interpolating polynomial efficiently?
- How accurate does the interpolating polynomial approximate a function?

Degree 0

The one-node table $\begin{array}{c|c} x & x_0 \\ \hline y & y_0 \end{array}$ is interpolated by the polynomial

$$p(x) = y_0.$$

- Is the polynomial unique?
- Would a polynomial of degree 1 that interpolates the table be uniquely determined?

Degree 1

The table $\begin{array}{c|cc} x & x_0 & x_1 \\ \hline y & y_0 & y_1 \end{array}$ is interpolated by the polynomial

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

Require: $x_0 - x_1 \neq 0$, i.e. the nodes are distinct.

Degree 1

The table $\begin{array}{c|cc} x & x_0 & x_1 \\ \hline y & y_0 & y_1 \end{array}$ is interpolated by the polynomial

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

Require: $x_0 - x_1 \neq 0$, i.e. the nodes are distinct.
This polynomial is of degree 1 (at most), and passes through both points

$$p(x_0) = \left(\frac{x_0 - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x_0 - x_0}{x_1 - x_0}\right) y_1 = \left(1\right) y_0 + \left(0\right) y_1 = y_0$$

$$p(x_1) = \left(\frac{x_1 - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x_1 - x_0}{x_1 - x_0}\right) y_1 = \left(0\right) y_0 + \left(1\right) y_1 = y_1.$$

Degree 1

The table $\begin{array}{c|cc} x & x_0 & x_1 \\ \hline y & y_0 & y_1 \end{array}$ is interpolated by the polynomial

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

Require: $x_0 - x_1 \neq 0$, i.e. the nodes are distinct.
This polynomial is of degree 1 (at most), and passes through both points

$$p(x_0) = \left(\frac{x_0 - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x_0 - x_0}{x_1 - x_0}\right) y_1 = \left(1\right) y_0 + \left(0\right) y_1 = y_0$$

$$p(x_1) = \left(\frac{x_1 - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x_1 - x_0}{x_1 - x_0}\right) y_1 = \left(0\right) y_0 + \left(1\right) y_1 = y_1.$$

Is the polynomial uniquely determined?

Degree 1

The table $\begin{array}{c|cc} x & x_0 & x_1 \\ \hline y & y_0 & y_1 \end{array}$ is interpolated by the polynomial

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

Require: $x_0 - x_1 \neq 0$, i.e. the nodes are distinct.
This polynomial is of degree 1 (at most), and passes through both points

$$p(x_0) = \left(\frac{x_0 - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x_0 - x_0}{x_1 - x_0}\right) y_1 = \left(1\right) y_0 + \left(0\right) y_1 = y_0$$

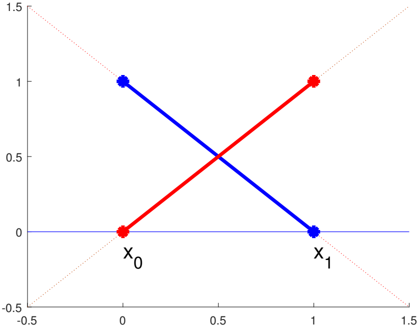
$$p(x_1) = \left(\frac{x_1 - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x_1 - x_0}{x_1 - x_0}\right) y_1 = \left(0\right) y_0 + \left(1\right) y_1 = y_1 .$$

Is the polynomial uniquely determined?
Yes! This p is called as **linear interpolation**.

Degree 1

$$p(x) = \left(\frac{x - x_1}{x_0 - x_1}\right) y_0 + \left(\frac{x - x_0}{x_1 - x_0}\right) y_1 = \lambda_{0,1}(x) y_0 + \lambda_{1,0}(x) y_1.$$

Here are two special polynomials of degree 1: $\lambda_{0,1}(x) = \left(\frac{x-x_1}{x_0-x_1}\right)$ and $\lambda_{1,0}(x) = \left(\frac{x-x_0}{x_1-x_0}\right)$.



They satisfy $\lambda_{0,1}(x_0) = 1$, $\lambda_{0,1}(x_1) = 0$, $\lambda_{1,0}(x_1) = 1$ and $\lambda_{1,0}(x_0) = 0$.

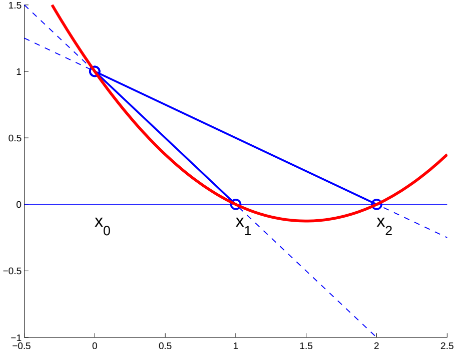
Degree 2

Consider the table

x	x_0	x_1	x_2
y	y_0	y_1	y_2

.

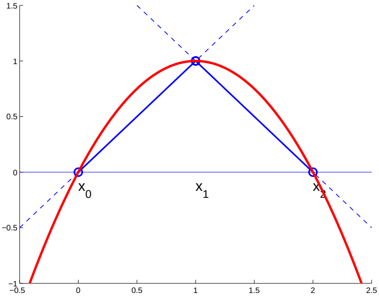
We can find a polynomial by multiplying two of our λ-polynomials, such that it equals 1 at one of x-values and 0 at any other x-values.

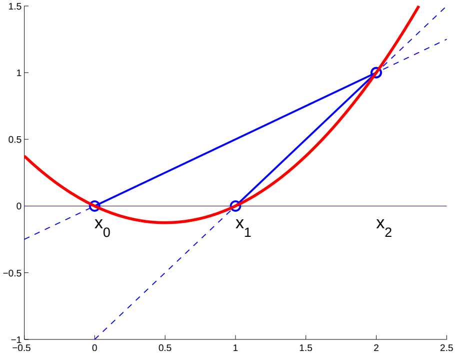


We call this polynomial $l_0(x) = \lambda_{0,1}(x)\lambda_{0,2}(x)$.

Degree 2

Similarly we can get another two polynomials:





$$l_1(x) = \lambda_{1,0}(x)\lambda_{1,2}(x) \quad \text{and} \quad l_2(x) = \lambda_{2,0}(x)\lambda_{2,1}(x)$$

Interpolation

We first define a system of special polynomials such that

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

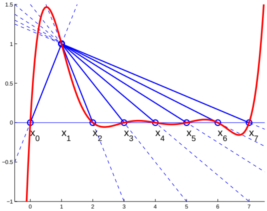
Then, the polynomial $p(x) = l_0(x)y_0 + l_1(x)y_1 + l_2(x)y_2$ is of at most second degree, and satisfies:

$$\begin{aligned} p(x_0) &= 1y_0 + 0y_1 + 0y_2 = y_0 \\ p(x_1) &= 0y_0 + 1y_1 + 0y_2 = y_1 \\ p(x_2) &= 0y_0 + 0y_1 + 1y_2 = y_2 \end{aligned}$$

Interpolating polynomial: Lagrange form

To interpolate a table of values

x	x_0	x_1	$\ldots$	x_n
y	y_0	y_1	$\ldots$	y_n



we first define **cardinal polynomials** l_i for $i = 0, \ldots, n$, which are

$$l_i(x) = \prod_{j=0,\ j\neq i}^{n} \lambda_{i,j}(x) = \prod_{j=0,\ j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j}\right),$$

where $\lambda_{i,j}$ satisfies $\lambda_{i,j}(x_i) = 1$ and $\lambda_{i,j}(x_j) = 0$. Hence, the cardinal polynomial l_i has the property

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

Interpolating polynomial: Lagrange form

To interpolate a table of values $\begin{array}{c|cccc} x & x_0 & x_1 & \dots & x_n \\ \hline y & y_0 & y_1 & \dots & y_n \end{array}$, the **Lagrange form of the interpolation polynomial** is

$$p_n(x) = \sum_{i=0}^{n} l_i(x) y_i.$$

Check:

$$p_n(x_j) = \sum_{i=0}^{n} l_i(x_j) y_i = 0 + \dots 0 + y_j + 0 + \dots + 0 = y_j.$$

We can **approximate any function**, f, by interpolating a table of its values.

Example

Approximate $f(x) = e^x \cos(x)$ (blue line) by finding a polynomial that interpolates the table:

x	0.0	1.0	1.5
$f(x)$	1.0	1.4687	0.3170

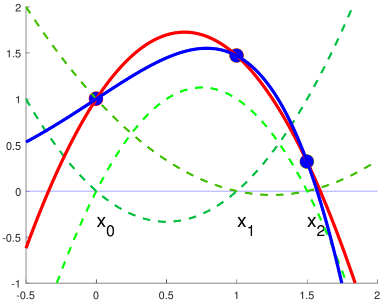
.

Cardinal polynomials are (dashed line):

$$l_0(x) = \left(\frac{x - 1.0}{0.0 - 1.0}\right)\left(\frac{x - 1.5}{0.0 - 1.5}\right)$$

$$l_1(x) = \left(\frac{x - 0.0}{1.0 - 0.0}\right)\left(\frac{x - 1.5}{1.0 - 1.5}\right)$$

$$l_2(x) = \left(\frac{x - 0.0}{1.5 - 0.0}\right)\left(\frac{x - 1.0}{1.5 - 1.0}\right).$$

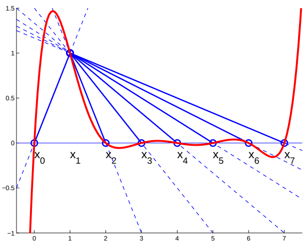


The approximation (red line) is

$$p_2(x) = 1.0\left(\frac{x - 1.0}{0.0 - 1.0}\right)\left(\frac{x - 1.5}{0.0 - 1.5}\right) + 1.4687\left(\frac{x - 0.0}{1.0 - 0.0}\right)\left(\frac{x - 1.5}{1.0 - 1.5}\right)$$
$$+ 0.3170\left(\frac{x - 0.0}{1.5 - 0.0}\right)\left(\frac{x - 1.0}{1.5 - 1.0}\right).$$

Interpolating polynomial: Lagrange form

Note that cardinal polynomials only depend on the values of the nodes (x-values), and not on y-values. For example, we show l_1 for a set of nodes $[0, 1, 2, 3, 4, 5, 6, 7]$ and $[2, 4, 2.5, 3, 1, 4.3, 4.6, 5]$.



The function values (y-values) are only used as weights for the cardinal polynomials in order to obtain the Lagrange form.

$$p_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i) \qquad \text{or} \qquad p_n(x) = \sum_{i=0}^{n} l_i(x) y_i .$$

Interpolating polynomial: Newton algorithm

We construct a sequence of polynomials, each of which interpolates one more pair of values in the table than the predecessor.

x	x_0	x_1	$\dots$	x_k	x_{k+1}	$\dots$	x_n
y	y_0	y_1	$\dots$	y_k	y_{k+1}	$\dots$	y_n

Assume that the polynomial p_k is of degree at most k and interpolates the first $k+1 < n$ pairs of values in the table, i.e. $p_k(x_i) = y_i$ for $i = 0, \dots, k$. Now we consider

$$P(x) = p_k(x) + c(x - x_0)(x - x_1)\cdots(x - x_k).$$

$P(x)$ is of degree at most $(k+1)$, and the constant c can be uniquely determined by

$$P(x_{k+1}) = p_k(x_{k+1}) + c(x_{k+1} - x_0)(x_{k+1} - x_1)\cdots(x_{k+1} - x_k) = y_{k+1}.$$

Then, we have $p_{k+1} = P$.

Example

Approximate $f(x) = e^x \cos(x)$ by finding a polynomial that interpolates the table:

x	0.0	1.0	1.5
$f(x)$	1.0	1.4687	0.3170

.

Example

Approximate $f(x) = e^x \cos(x)$ by finding a polynomial that interpolates the table:

x	0.0	1.0	1.5
$f(x)$	1.0	1.4687	0.3170

.

Start with $k = 0$, i.e. we use a degree 0 polynomial p_0 to interpolate the first datum. Obviously, we have

$$p_0(x) = y_0 = 1.0.$$

Example

Approximate $f(x) = e^x \cos(x)$ by finding a polynomial that interpolates the table:

x	0.0	1.0	1.5
$f(x)$	1.0	1.4687	0.3170

.

Start with $k = 0$, i.e. we use a degree 0 polynomial p_0 to interpolate the first datum. Obviously, we have

$$p_0(x) = y_0 = 1.0.$$

Now, we need find p_1 to interpolate the first two data, and p_1 is in the form

$$p_1(x) = p_0(x) + c(x - 0.0) = 1.0 + c(x - 0.0).$$

According to $p_1(1.0) = 1.0 + c(1.0 - 0.0) = 1.4687$, we obtain $c = 0.4687$, i.e.,

$$p_1(x) = 1.0 + 0.4687(x - 0.0).$$

Example

Approximate $f(x) = e^x \cos(x)$ by finding a polynomial that interpolates the table:

x	0.0	1.0	1.5
$f(x)$	1.0	1.4687	0.3170

.

After obtain

$$p_1(x) = 1.0 + 0.4687(x - 0.0).$$

Now, we need find p_2 to interpolate all three data, and p_2 is in the form

$$p_2(x) = p_1(x) + c(x-0.0)(x-1.0) = 1.0 + 0.4687(x-0.0) + c(x-0.0)(x-1.0).$$

According to

$$p_2(1.5) = 1.0 + 0.4687(1.5 - 0.0) + c(1.5 - 0.0)(1.5 - 1.0) = 0.3170,$$

we obtain $c = -1.8481$, i.e.,

$$p_2(x) = 1.0 + 0.4687(x - 0.0) - 1.8481(x - 0.0)(x - 1.0).$$

Existence and uniqueness of polynomial interpolation

Theorem: *If points $x_0, x_1, \ldots, x_n$ are distinct, then for arbitrary real values $y_0, y_1, \ldots, y_n$, there is a unique polynomial p of degree at most n such that $p(x_i) = y_i$ for $i = 0, 1, \ldots, n$.*

Proof of existence (inductive reasoning):

Start with $n = 0$: The pair of values (x_0, y_0) can be uniquely interpolated by the 0-degree polynomial $p_0(x) = y_0$.

Induction stage: Based on the Newton algorithm, we can uniquely determine a constant c, which leads to a polynomial of degree at most n.

Uniqueness: If both polynomials p and q interpolate the table, then the polynomial $p - q$ is of degree at most n and takes the value 0 at all $n + 1$ distinct points $x_0, x_1, \ldots, x_n$. However, a nonzero polynomial of degree n can have at most n roots. Therefore, we conclude that $p = q$. $\square$

The Newton form and Lagrange form are just two different derivations for precisely the same polynomial.

Nested multiplication

Based on the Newton algorithm we can rewrite the interpolating polynomial by "nested multiplication".

$$
\begin{aligned}
p_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots \\
&\quad + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\
&= a_0 + (x - x_0)\Big(a_1 + a_2(x - x_1) + a_3(x - x_1)(x - x_2) + \cdots \\
&\quad + a_n(x - x_1)(x - x_2)\cdots(x - x_{n-1})\Big) \\
&= a_0 + (x - x_0)\Big(a_1 + (x - x_1)\Big(a_2 + a_3(x - x_2) + \cdots \\
&\quad + a_n(x - x_2)\cdots(x - x_{n-1})\Big)\Big) \\
&\vdots \\
&= a_0 + (x - x_0)\Big(a_1 + (x - x_1)\Big(a_2 + \cdots \\
&\quad + (x - x_{n-2})(a_{n-1} + (x - x_{n-1})(a_n)\Big)\ldots\Big)\Big)
\end{aligned}
$$

FLOPS for evaluating interpolation polynomial

The Lagrange form consists of $n+1$ cardinal polynomials of degree n.

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

To evaluate each cardinal polynomial, we need $2n$ subtractions, n divisions and $n-1$ multiplications. So to evaluate the whole Lagrange form we need $4n(n+1)+n$ flops (floating point operations).

Newton algorithm provides interpolation polynomials in a nested multiplication, which need much less flops for evaluation.

Nested multiplication only requires $3n$ flops.

Vandermonde matrix

Another way of finding a polynomial $p_n(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$ to interpolate a table is to solve $n+1$ equations, $p_n(x_i) = y_i$, with $n+1$ unknown coefficients $c_0, c_1, \ldots, c_n$. This can be written as a linear system:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- The coefficient matrix is called a Vandermonde matrix.
- For $n \approx 10$ it can already have numerical issue to solve it. (We will be back to this problem in the last block of the course.)

Inverse interpolation

If we reverse the arguments x and $y = f(x)$ in the table for interpolation, then we would obtain an approximation, $x = p(y)$, of the inverse function $f^{-1}(y)$.

- Inverse interpolation is often used to approximate an inverse function.
- Inverse interpolation can be used to find a root of the given function: We create a table of values $(f(x_i), x_i)$ and interpolate with a polynomial, p. Thus, we obtain $p(y_i) = x_i$, and the approximate root is then given by $p(0)$. (Note: the points x_i should be chosen near the unknown root.)

Errors in polynomial interpolation
Chapter 4.2 until Theorem 2

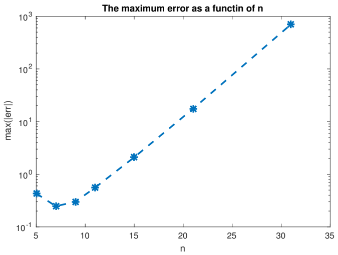
A problem with high-degree polynomials: A polynomial of degree n has n zeros, i.e, its curve crosses the x-axis n times, which results in wild oscillations!

Example: Let p_n be the polynomial that interpolates the function

$$f(x) = \frac{1}{1+x^2}$$

at $n+1$ equally spaced points on the interval $[-5, 5]$. Then, we can prove that

$$\lim_{n\to\infty} \max_{-5\le x\le 5} \left| f(x) - p_n(x) \right| = \infty.$$



First interpolation error theorem

Can we assess when interpolation works well?

First interpolation error theorem

Can we assess when interpolation works well?
- **Yes**! There are two results on assessing the errors of interpolation.

First interpolation error theorem

Can we assess when interpolation works well?
- **Yes**! There are two results on assessing the errors of interpolation.

> *First Interpolation Error Theorem: If p_n is the polynomial of degree at most n that interpolates f at the $n+1$ distinct nodes $x_0, x_1, \ldots, x_n \in [a,b]$ and if $f^{(n+1)}$ is continuous, then for each $x \in [a,b]$, there is a $\xi \in (a,b)$ for which*
>
> $$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i).$$

Proof of first interpolation error theorem

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i)$$

Obviously, the equation is valid for all nodes. Now, we need show that the equation is also valid for the case that x **is not a node**.

Proof of first interpolation error theorem

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i)$$

Obviously, the equation is valid for all nodes. Now, we need show that the equation is also valid for the case that x **is not a node**.

Define $\omega(t) = \prod_{i=0}^{n} (t - x_i)$, which equals to zero at all nodes x_0, x_1, $\ldots$, x_n, and introduce a constant c as

$$c = \frac{f(x) - p_n(x)}{\omega(x)},$$

which is well defined because $\omega(x) \neq 0$.

Proof of first interpolation error theorem - continue

Define a function in the variable t:

$$\varphi(t) = f(t) - p_n(t) - c\omega(t).$$

Note that $f(t) - p_n(t)$ and $\omega(t)$ equal 0 at all nodes. Hence, $\varphi(t) = f(t) - p_n(t) - c\omega(t)$ also takes 0 at all nodes. Moreover, according to the definition of c, $\varphi(t)$ equals to 0 at $t = x$. That is:

- φ takes the value 0 at the $n+2$ points $x_0, x_1, \ldots, x_n$ and x.
- Rolle's Theorem states that $\varphi' = 0$ for at least $n+1$ points between $x_0, x_1, \ldots, x_n$ and x.
- Rolle's Theorem states that $\varphi'' = 0$ for at least n points.
- $\varphi''' = 0$ at $n-1$ points.
- $\vdots$
- Rolle's Theorem states that $\varphi^{(n+1)} = 0$ for at least one point $\xi \in (a, b)$, that is:

Proof of first interpolation error theorem - continue

$$0 = \varphi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p_n^{(n+1)}(\xi) - c\omega^{(n+1)}(\xi).$$

In this equation, we know that:

- $\omega(t) = \prod_{i=0}^{n}(t - x_i) = t^{(n+1)} + (\text{lower-order terms in } t)$
- So $\omega^{(n+1)}(t) = \omega^{(n+1)}(\xi) = (n+1)!$
- p_n is a polynomial of degree $\leq n$, so $p_n^{(n+1)}(\xi) = 0$.

Thus, we have:

$$\begin{aligned} 0 = \varphi^{(n+1)}(\xi) &= f^{(n+1)}(\xi) - 0 - c(n+1)! \\ &= f^{(n+1)}(\xi) - \frac{f(x) - p_n(x)}{\omega(x)}(n+1)! \end{aligned}$$

$$\Updownarrow$$

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n}(x - x_i)$$

$\square$

Second interpolation error theorem

Often the interpolation nodes are equally spaced in $[a, b]$, so $x_{i+1} - x_i = h = (b-a)/n$. Then, we can obtain that for any $x \in [a, b]$

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{1}{4} h^{n+1} n!.$$

Thus, we have

$$\left| f(x) - p_n(x) \right| = \frac{1}{(n+1)!} \left| f^{(n+1)}(\xi) \right| \prod_{i=0}^{n} |x - x_i| \leq \frac{1}{4(n+1)} \left| f^{(n+1)}(\xi) \right| h^{n+1}.$$

Moreover, if $\left| f^{(n+1)}(x) \right| \leq M$ for all $x \in [a, b]$, then we find a bound independent from x:

Second interpolation error theorem

Second Interpolation Error Theorem: Let f be a function such that $f^{(n+1)}$ is continuous on $[a,b]$ and satisfies $\left|f^{(n+1)}(x)\right| \leq M$. Let p_n be the polynomial of degree at most n that interpolates f at the $n+1$ equally spaced nodes in $[a,b]$, including the endpoints. Then, for all $x \in [a,b]$, we have

$$\left|f(x) - p_n(x)\right| \leq \frac{1}{4(n+1)}\left|f^{(n+1)}(\xi)\right| h^{n+1} \leq \frac{1}{4(n+1)} M h^{n+1},$$

where $h = (b-a)/n$ is the spacing between nodes.

Second interpolation error theorem - example 1

Interpolate exponential function $f(x) = e^x$ in $[0, 2]$. We have $f(x) = f'(x) = f''(x) = \cdots = f^{(k)}(x) = e^x$ and

$$\left| f^{(k)}(x) \right| = e^x \leq e^2 < 7.4 = M,$$

which is independent on k and x.
An interpolation polynomial with $n+1$ equally spaced nodes in $[0, 2]$ satisfies

$$\left| f(x) - p_n(x) \right| \leq \frac{1}{4(n+1)} M h^{n+1} = \frac{1}{4(n+1)} 7.4 \left(\frac{2}{n} \right)^{n+1}.$$

If $n = 10$, the error $\left| f(x) - p_n(x) \right| \leq 3.5 \times 10^{-9}$.

Due to *small derivative* of the exponential function, the polynomial interpolation works well here. But with a large n cancellation error becomes dominated (loss of significance in subtraction).

Second interpolation error theorem - example 2

Interpolate the function $f(x) = x^{-1}$ in $[0.1, 1.1]$. We have

$$\begin{aligned} f'(x) &= (-1)x^{-2} \\ f''(x) &= (-1)(-2)x^{-3} \\ &\vdots \\ \left|f^{n+1}(x)\right| &= (n+1)!\left|x\right|^{-(n+2)} \leq (n+1)!0.1^{-(n+2)} = (n+1)!10^{n+2}. \end{aligned}$$

Based on the second interpolation error theorem, we obtain:

$$\left|f(x) - p_n(x)\right| \leq \frac{1}{4(n+1)} M h^{n+1} = \frac{1}{4(n+1)}(n+1)!10^{n+2}\left(\frac{1}{n}\right)^{n+1}$$

If $n = 10$, then the upper bound on the interpolation error is
$\frac{10!}{4}10 = 9,072,000!$
The reason is that *the high-order derivative increases too fast*.

Summary

Consider a table of values:

x	x_0	x_1	$\ldots$	x_n
y	y_0	y_1	$\ldots$	y_n

We assume that x_i's are $n+1$ distinct points, i.e. $x_i \neq x_j$ if $i \neq j$.

We can find the interpolating polynomial of degree at most n by applying Lagrange form or Newton algorithm.

Based on Newton algorithm we can rewrite the interpolating polynomial by "nested multiplication".

Summary

Consider a table of values:

x	x_0	x_1	$\dots$	x_n
y	y_0	y_1	$\dots$	y_n

We assume that x_i's are $n+1$ distinct points, i.e. $x_i \neq x_j$ if $i \neq j$.

We can find the interpolating polynomial of degree at most n by applying Lagrange form or Newton algorithm.

Based on Newton algorithm we can rewrite the interpolating polynomial by "nested multiplication".

A function f can be approximated by interpolation of $f(x_i) = y_i$, $i = 0, \dots, n$.

The upper bound of the error for interpolation can be estimated by first or second interpolation error theorem.

Approximation by interpolation doesn't always work well. It requires that "high-order derivative" of f has small value.

Mathematical overview (not in the book)

- Consider a fixed selected set of nodes x_i, for $i = 0, \dots, n$.
- $p_{1,n}$ approximates the function f_1: $p_{1,n}(x_i) = f_1(x_i)$ for $i = 0, \dots, n$.
- $p_{2,n}$ approximates the function f_2: $p_{2,n}(x_i) = f_2(x_i)$ for $i = 0, \dots, n$.

Mathematical overview (not in the book)

- Consider a fixed selected set of nodes x_i, for $i = 0, \ldots, n$.
- $p_{1,n}$ approximates the function f_1: $p_{1,n}(x_i) = f_1(x_i)$ for $i = 0, \ldots, n$.
- $p_{2,n}$ approximates the function f_2: $p_{2,n}(x_i) = f_2(x_i)$ for $i = 0, \ldots, n$.
- Let $a, b \in \mathbb{R}$. Which polynomial approximates $a\, f_1 + b\, f_2$?

Mathematical overview (not in the book)

- Consider a fixed selected set of nodes x_i, for $i = 0, \ldots, n$.
- $p_{1,n}$ approximates the function f_1: $p_{1,n}(x_i) = f_1(x_i)$ for $i = 0, \ldots, n$.
- $p_{2,n}$ approximates the function f_2: $p_{2,n}(x_i) = f_2(x_i)$ for $i = 0, \ldots, n$.
- Let $a, b \in \mathbb{R}$. Which polynomial approximates $a\,f_1 + b\,f_2$?
- The answer is $a\,p_{1,n} + b\,p_{2,n}$, because

Mathematical overview (not in the book)

- Consider a fixed selected set of nodes x_i, for $i = 0, \ldots, n$.
- $p_{1,n}$ approximates the function f_1: $p_{1,n}(x_i) = f_1(x_i)$ for $i = 0, \ldots, n$.
- $p_{2,n}$ approximates the function f_2: $p_{2,n}(x_i) = f_2(x_i)$ for $i = 0, \ldots, n$.
- Let $a, b \in \mathbb{R}$. Which polynomial approximates $a\, f_1 + b\, f_2$?
- The answer is $a\, p_{1,n} + b\, p_{2,n}$, because

$$a\, p_{1,n}(x_i) + b\, p_{2,n}(x_i) = a\, f_1(x_i) + b\, f_2(x_i), \text{for } i = 0, \ldots, n,$$

  and this polynomial is uniquely determined.

Hence, polynomial interpolation can be considered as a **linear mapping** from a function space to the space of polynomials.

Mathematical overview

Corollary The monomials x^i, $i = 0, \ldots, n$, are linear independent functions.

Proof: Assume a linear combination of the monomials equals to 0, i.e,

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = 0 \quad \text{for all } x.$$

- When $n = 1$, we have $a_0 + a_1 x = 0$ for all x. Set $x = 0$, we obtain $a_0 = 0$. Then, from $a_1 x = 0$, obviously we have $a_1 = 0$.
- When $n = 2$, we set $x = 0$ and obtain $a_0 = 0$. Then, we have $x(a_1 + a_2 x) = 0$ for all x. Hence, $a_1 + a_2 x$ must be 0, and $a_1 = a_2 = 0$ according to the case $n = 1$.
- $\cdots\cdots$

In a conclusion, the coefficients $(a_0, a_1, \ldots, a_n)$ have to be 0 for any integer n.